НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.582

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Модифікований підхід використання WebView для розроблення мобільних додатків»

Виконала:
студентка ІІ курсу, групи КП-81мп
Родіонова Віра Олексіївна _____

Керівник:
Ст. викладач кафедри ПЗКС, к.т.н.,
Люшенко Л.А. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай М.В. _____

Рецензент:
Доц. кафедри ММСА ІПСА, к.ф-м.н., доц.,
Шубенкова І.А. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (освітня програма) – 121 «Інженерія програмного забезпечення» ("Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем")

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студентці

Родіоновій Вірі Олексіївні

1. Тема дисертації «Модифікований підхід використання WebView для розроблення мобільних додатків», науковий керівник дисертації Люшенко Леся Анатоліївна, к.т.н., затверджені наказом по університету від «13» листопада 2019 р. № 3895-С.

2. Термін подання студентом дисертації «16» грудня 2019 р.

3. Об'єкт дослідження: фреймворки для створення крос-платформних користувацьких додатків.

4. Предмет дослідження: способи інтеграції з "рідними" API мобільних платформ та архітектура.

5. Перелік завдань, які потрібно розробити:
- оцінити сучасний стан проблеми, обгрунтувати актуальність напрямку досліджень, сформулювати мету та задачі дослідження;
- описати сферу розповсюдження крос-платформних користувацьких додатків;
- сформувати вимоги до модифікованого способу використання WebView під час аналізу існуючих рішень для крос-платформної розробки користувацьких додатків;
- описати принципи, що використовуються у модифікованому способі використання WebView: описати їх особливості та детально описати ідею структурних компонент системи;
- описати структурні компоненти, що є складовими модифікованого спосіб використання WebView для розробки користувацьких мобільних додатків;
- виконати тестування розробленого програмного забезпечення модифікованого способу використання WebView для створення мобільних додатків та проаналізувати отримані результати;
- описати бізнес-модель, що дозволить представити на ринку повноцінний програмний продукт із використанням представлених у роботі напрацювань.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
   - загальна архітектура підходу;
   - схема організації крос-платформного тексту програми у розроблюваній системі;
   - діаграма класів каналу комунікації;
   - результати аналізу розробленого програмного забезпечення;
   - дерево проблем та рішень.

7. Орієнтовний перелік публікацій:
   - Тези доповіді “Спосіб використання WEBVIEW для розроблення мобільних додатків”

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., к.т.н., доцент		

9. Дата видачі завдання «25» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.11.2018	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.12.2018	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.02.2019	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	05.04.2019	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.05.2019	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	15.06.2019	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2019	13.11.2019	
8.	Оформлення текстової і графічної частини магістерської дисертації	05.12.2019	

Студент В.О. Родіонова

Науковий керівник дисертації Л.А. Люшенко

РЕФЕРАТ

Актуальність теми. Сьогодні для успішного ведення бізнесу, що зорієнтований на взаємодію з великою кількістю людей, необхідно або створювати власні додатки, або ж інтегруватися в існуючі системи такі, як соціальні мережі, конструктори сайтів. Створювання сучасних програмних систем вимагає розповсюдження на найбільшу користувацьку аудиторію, що складається з веб-додатку, мобільних додатків для iOS та Android. Більшість сучасних платформ для крос-платформної розробки використовують системний компонент WebView для створення додатків, але вони мають свої обмеження, а саме:

- створення кінцевих збірок користувацьких додатків відбувається на віддаленому сервері і розробник не має доступу до вихідного тексту програми;
- підтримують малу кількість цільових платформ, а створення додаткових модулів взаємодії із системними сервісами вимагає специфічних знань цільової платформи.

Відповідно до цього актуальним є створення модифікованого методу роботи з WebView для розроблення мобільних додатків, який дозволить легко інтегруватися в існуючі платформи та розробляти користувацькі додатки без специфічних знань цільових платформ.

Об’єктом дослідження є фреймворки для створення крос-платформних користувацьких додатків.

Предметом дослідження є способи інтеграції з "рідними" API мобільних платформ та архітектура.

Мета роботи: дослідження існуючих рішень для крос-платформної розробки користувацьких додатків, розроблення способу використання WebView для написання крос-платформних користувацьких додатків та

розроблення архітектури на основі модульного підходу проектування систем.

Методи дослідження. В даній роботі використовуються методи теоретичного дослідження: аналіз, синтез та узагальнення. Також застосовувалися емпіричні методи: експеримент, спостереження, вимірювання та опис.

Наукова новизна роботи полягає в наступному:

1. Розроблено канал комунікації між React Native та WebView, що базується на основі клієнт-серверної архітектури та вбудованих механізмах глобальних генераторів подій у браузері. Канал комунікації має декілька складових таких як, Клієнт, Сервер, Middleware, класи-обгортки для глобальних генераторів подій на обох сторонах та класи, які поєднали в собі всі складові.

2. Стандартизовано взаємодію із системними сервісами та розроблено плагіни, що забезпечують доступ до платформо-залежних API. Вони надають однаковий інтерфейс, незалежно від того, на якій з платформ зараз запущено додаток. Тим самим звільняючи розробника від перевірок на кожному кроці типу поточного середовища виконання.

3. Розроблено менеджер плагінів, який є відповідальним за роботу з різними типами плагінів (утилітами й компонентами), їх зберігання у реєстрі та забезпечення доступу до них.

Практична цінність отриманих в роботі результатів полягає в тому, що розроблену систему можна використовувати для створення платформо-орієнтованих користувацьких додатків із спільною кодовою базою, яка дозволяє: зробити розроблення користувацьких додатків більш гнучким; скоротити витрати грошових та управлінських ресурсів на велику команду розробників. Через використання React Native в якості базової платформи, за допомогою розробленої системи можна створити додаток, який буде зібрано на більше ніж 5 платформ.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на XII науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2019 (Київ, 13-15 листопада 2019 р.).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п'яти розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи.

У першому розділі було проаналізовано існуючі рішення для створення крос-платформних додатків. Виявлено їх переваги та недоліки. Досліджено методи досягнення крос-платформності. На основі досліджень та результатів аналізу було сформовано вимоги до розроблюваного способу модифікації, а саме: розроблення крос-платформного користувацького додатка на основі React Native з можливістю створювати збірки на більше ніж 5 платформ за допомогою middleware й стандартизації платформо-орієнтованого коду, та розроблення програмної системи, що реалізує запропонований спосіб модифікації.

У другому розділі було детально викладено рішення, що пропонується. Модифікація підходу полягає у тому, що у розроблюваній системі було використано допоміжний рівень у вигляді React Native. Також модифікацією можна вважати сприйняття веб-середовища, як ще однієї цільової платформи. Адже, взаємодія із платформо-орієнтованою частиною тексту програми відбувається у тому ж процесі, а не через допоміжний канал комунікації, а така взаємодія має відмінності від стандартного підходу.

У третьому розділі було описано розроблені системні модулі, а саме: класи, що складають канал комунікації між сторонами веб та React Native;

особливості використання платформного компонента відображення веб-вмісту сторінок – WebView; плагіни – архітектурні компоненти, що відповідають за стандартизацію API взаємодії тексту програми користувацького додатка із системними сервісами; класи архітектури для централізованого доступу до плагінів на обох сторонах системи (веб та React Native). Завдяки їх взаємодії стало можливим, щоб веб-додаток працював із платформними API, які надають операційні системи смартфонів, як з "рідними" мобільними додатками, та не втратити сумісність із веб-середовищем.

У четвертому розділі було описано методологію тестування для розробленої системи, проаналізовано результати роботи та визначено напрями для подальшого вдосконалення системи у майбутньому. За результати тестування можна стверджувати, що розроблена система не має помилок, які можуть призвести до виключних ситуацій при роботі користувача. Тому може вважатися готовою до залучення бета-тестувальників.

У п’ятому розділі було сформовано та побудовано бізнес-модель стартапу для кінцевого продукту, що описує ключові моменти в організації діяльності, пов’язаної з поширенням розробленої системи для створення крос-платформних користувацьких додатків.

У висновках проаналізовано отримані результати роботи.

У додатках наведено допоміжні функції для маніпулювання внутрішнім станом плагінів, приклади специфічних для платформи плагінів.

Робота виконана на 82 аркушах, містить 3 додатки та посилання на список використаних літературних джерел з 27 найменувань. У роботі наведено 19 рисунків, 9 таблиць та 7 лістингів.

Ключові слова: крос-платформна розробка, WebView, React Native, mole-rpc, користувацькі додатки.

ABSTRACT

Actuality of theme. Today, in order to successfully run a business that is focused on interacting with a large number of people, you need to either create your own applications or integrate into existing systems such as social networks, site designers. Creating state-of-the-art software systems requires distribution to the largest user audience, consisting of a web application, mobile applications for iOS and Android. Most current cross-platform development platforms use the WebView system component to build applications, but they have their limitations, namely:

- the creation of the final user application builds takes place on a remote server and the developer does not have access to the program's source text;
- support a small number of target platforms, and the creation of additional modules for interaction with system services requires specific knowledge of the target platform.

Accordingly, it is important to create a modified method of working with WebView for mobile application development, which will make it easy to integrate into existing platforms and develop custom applications without specific knowledge of the target platforms.

Object of research is frameworks for cross-platform custom applications.

Subject of research is how to integrate with the native APIs of mobile platforms and architecture.

Research objective is to explore existing cross-platform custom application development solutions, develop a way to use WebView to write cross-platform custom applications, and develop an architecture based on a modular system design approach.

Research methods. This paper uses methods of theoretical research: analysis, synthesis and generalization. Empirical methods were also used: experiment, observation, measurement and description.

The **scientific novelty** of the work is as follows:

1. A communication channel between React Native and WebView was developed, based on client-server architecture and the built-in mechanisms of global event generators in the browser. The communication channel has several components such as, Client, Server, Middleware, wrapper classes for global event generators on both sides, and classes that combine all components.
2. Standardized interaction with system services and plugins that provide access to platform-dependent APIs. They provide the same interface, regardless of which platform is currently running the application. Thus freeing the developer from checks at each step of the current runtime type.
3. A plug-in manager has been developed that is responsible for handling various types of plug-ins (utilities and components), storing them in the registry and providing access to them.

The practical value of the results obtained is that the developed system can be used to create platform-oriented user applications with a common codebase, which allows: to make the development of user applications more flexible; reduce the cost of money and management resources for a large team of developers. By using React Native as the base platform, you can use the developed system to create an application that will build on more than 5 platforms.

Approbation. The main provisions and results of the work were presented and discussed at the XII Scientific Conference of Undergraduate and Graduate Students in Applied Mathematics and Computing PMK-2019 (Kyiv, November 13-15, 2019).

Structure and content of the thesis. The master's thesis consists of an introduction, five sections, conclusions and appendices.

The introduction gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the research, shows the scientific novelty of the obtained results and practical value of the work.

The first section analyzes existing solutions for cross-platform application creation. Their advantages and disadvantages are revealed. The methods of cross-platform achievement are investigated. Based on the research and analysis results, requirements for the modification modality were developed, namely: development of a cross-platform user application based on React Native with the ability to build assemblies on more than 5 platforms using middleware and standardization of platform-oriented code, and development of a software system that implements the proposed modification method.

The second section details the proposed solution. A modification of the approach is that the assistive system in the form of React Native was used in the developed system. The perception of the web environment as another target platform can also be considered a modification. After all, the interaction with the platform-oriented part of the program text takes place in the same process, not through an auxiliary communication channel, and such interaction is different from the standard approach.

The third section describes the developed system modules, namely: the classes that make up the communication channel between the Web and React Native parties; features of the WebView web content rendering platform component; plugins - architectural components that are responsible for standardizing the API of the user application text interaction with system services; architecture classes for centralized access to plugins on both sides of the system (web and React Native). Through their interaction, it has become possible for the web application to work with platform APIs that provide operating

systems of smartphones, as with "native" mobile applications, and not lose compatibility with the web environment.

The fourth section describes the testing methodology for the developed system, analyzed the results of the work and identified directions for further improvement of the system in the future. The test results indicate that the system developed does not have errors that can lead to exceptional situations when the user is operating. Therefore, it may be considered ready to engage beta testers.

The fifth section builds and builds a business model for a startup for the end product that describes the key issues in organizing the expansion of a cross-platform custom application development system.

The conclusion is analyzed the results of this master's work.

The applications provide auxiliary functions for manipulating the internal state of plugins, examples of platform-specific plugins.

The work is made on 82 sheets, contains 3 appliccations and links to the list of used literary sources of 27 titles. The paper presents 19 figures, 9 tables and 7 listings.

Keywords: cross-platform development, WebView, React Native, mole-rpc, custom applications.

РЕФЕРАТ

Актуальность темы. Сегодня для успешного ведения бизнеса, ориентированный на взаимодействие с большим количеством людей, необходимо либо создавать собственные приложения, или же интегрироваться в существующие системы такие, как социальные сети, конструкторы сайтов. Созидания современных программных систем требует распространения на самую пользовательскую аудиторию, состоящую из веб-приложения, мобильных приложений для iOS и Android. Большинство современных платформ для кросс-платформенной разработки используют системный компонент WebView для создания приложений, но они имеют свои ограничения, а именно:

- создание конечных сборников пользовательских приложений происходит на удаленном сервере и разработчик не имеет доступа к исходному тексту программы;
- поддерживают малое количество целевых платформ, а создание дополнительных модулей взаимодействия с системными сервисами требует специфических знаний целевой платформы.

В соответствии с этим актуальным является создание модифицированного метода работы с WebView для разработки мобильных приложений, который позволит легко интегрироваться в существующие платформы и разрабатывать пользовательские приложения без специфических знаний целевых платформ.

Объектом исследования является фреймворки для создания кросс-платформенных пользовательских приложений.

Предметом исследования являются способы интеграции с "родными" API мобильных платформ и архитектура.

Цель работы: исследование существующих решений для кросс-платформенной разработки пользовательских приложений, разработка

способа использования WebView для написания кросс-платформенных пользовательских приложений и разработка архитектуры на основе модульного подхода проектирования систем.

Методы исследования. В данной работе используются методы теоретического исследования: анализ, синтез и обобщение. Также применялись эмпирические методы: эксперимент, наблюдение, измерение и описание.

Научная новизна работы заключается в следующем:

1. Разработаны канал коммуникации между React Native и WebView, что базируется на основе клиент-серверной архитектуры и встроенных механизмах глобальных генераторов событий в браузере. Канал коммуникации имеет несколько составляющих таких как, Клиент, Сервер, middleware, классы-обертки для глобальных генераторов событий на обеих сторонах и классы, которые объединили в себе все составляющие.
2. Стандартизирована взаимодействие с системными сервисами и разработаны плагины, обеспечивающие доступ к платформо-зависимых API. Они предоставляют одинаковый интерфейс, независимо от того, на какой из платформ сейчас запущен приложение. Тем самым освобождая разработчика от проверок на каждом шагу типа текущей среды выполнения.
3. Разработаны менеджер плагинов, который отвечает за работу с различными типами плагинов (утилитами и компонентами), их хранения в реестре и обеспечения доступа к ним.

Практическая ценность полученных в работе результатов заключается в том, что разработанную систему можно использовать для создания платформенно-ориентированных пользовательских приложений с общей кодовым базой, позволяет сделать разработку пользовательских приложений более гибким; сократить расходы денежных и управленческих

ресурсов на большую команду разработчиков. Из-за использования React Native в качестве базовой платформы, с помощью разработанной системы можно создать приложение, которое будет собрано на более 5 платформ.

Апробация. Основные положения и результаты работы были представлены и обсуждались на ХИИ научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2019 (Киев, 13-15 ноября 2019).

Структура и объем работы. Магистерская диссертация состоит из введения, пяти глав, заключения и приложений.

Во введении дано общая характеристика работы, выполнена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов и практическую ценность работы.

В первом разделе были проанализированы существующие решения для создания кросс-платформенных приложений. Выявлены их преимущества и недостатки. Исследованы методы достижения кросс-платформности. На основе исследований и результатов анализа был сформирован требования к разрабатываемого способа модификации, а именно: разработка кросс-платформенного пользовательского приложения на основе React Native с возможностью создавать сборки на более 5 платформ с помощью middleware и стандартизации платформо-ориентированного кода, и разработка программной системы , реализующего предложенный способ модификации.

Во втором разделе детально изложены решения, предлагается. Модификация подхода заключается в том, что в разрабатываемой системе было использовано вспомогательный уровень в виде React Native. Также модификацией можно считать восприятия веб-среды, как еще одной целевой платформы. Ведь, взаимодействие с платформо-ориентированной

частью текста программы происходит в том же процессе, а не через вспомогательный канал коммуникации, а такое взаимодействие имеет отличия от стандартного подхода.

В третьем разделе описано разработаны системные модули, а именно: классы, составляющих канал коммуникации между сторонами веб и React Native; особенности использования платформенного компонента отображения веб-содержимого страниц - WebView; плагины - архитектурные компоненты, отвечающие за стандартизацию API взаимодействия текста программы пользовательского приложения с системными сервисами; классы архитектуры для централизованного доступа к плагинам на обеих сторонах системы (веб и React Native). Благодаря их взаимодействия стало возможным, чтобы веб-приложение работал с платформенными API, которые предоставляют операционные системы смартфонов, как с "родными" мобильными приложениями, и не потерять совместимость с веб-средой.

В четвертом разделе описано методологию тестирования для разработанной системы, проанализированы результаты работы и определены направления для дальнейшего совершенствования системы в будущем. По результатам тестирования можно утверждать, что разработанная система не имеет ошибок, которые могут привести к исключений при работе пользователя. Поэтому может считаться готовой к привлечению бета-тестировщиков.

В пятом разделе было сформировано и построено бизнес-модель стартапа для конечного продукта, описывает ключевые моменты в организации деятельности, связанной с распространением разработанной системы для создания кросс-платформенных пользовательских приложений.

В выводах проанализированы полученные результаты работы.

В приложениях приведены вспомогательные функции для манипулирования внутренним состоянием плагинов, примеры специфических для платформы плагинов.

Работа выполнена на 82 листах, содержит 3 приложения и ссылки на список использованных литературных источников из 27 наименований. В работе приведены 19 рисунков, 9 таблиц и 7 листингов.

Ключевые слова: кросс-платформенная разработка, WebView, React Native, mole-rpc, пользовательские приложения.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

React Native – це JavaScript фреймворк для написання реальних мобільних додатків для iOS та Android. Він базується на React, бібліотеці JavaScript Facebook для створення інтерфейсів користувача, але замість націлення на браузер, він націлений на мобільні платформи [1].

WebView – це вбудований веб-переглядач, який може використовувати вбудована програма для відображення веб-вмісту [2].

JSCore – це рамка, яка забезпечує двигун JavaScript для реалізації WebKit і забезпечує цей тип сценаріїв в інших контекстах в macOS [3].

Пропрієтарне програмне забезпечення – (від англ. proprietary software) – це програмне забезпечення, на яке зберігаються як немайнові, так і майнові авторські права.

Фреймворк – це реальна або концептуальна структура, призначена слугувати опорою або керівництвом для побудови чогось, що розширює структуру на щось корисне.

API (Application Programming Interface) – це набір публічних методів та властивостей, які використовуються для взаємодії з іншими об’єктами у програмі.

ВСТУП

Сьогодні для успішного ведення бізнесу, що зорієнтований на взаємодію з великою кількістю людей, необхідно або створювати власні додатки, або ж інтегруватися в існуючі системи такі, як соціальні мережі, конструктори сайтів. Створювання сучасних програмних систем вимагає розповсюдження на найбільшу користувацьку аудиторію, що складається з веб-додатку, мобільних додатків для iOS та Android.

Бізнеси зацікавлені у створенні та утриманні великої кількості постійних клієнтів. Тобто, утриманні високого рівня лояльності та наданні якісного інструменту для вирішення проблем клієнтів – користувачів електронних додатків.

У сфері інформаційних технологій існує конкуренція. І як наслідок існують однотипні платформи, які вимагають адаптуватися під їх специфічні стандарти. Тому виникають проблеми пов’язані із створенням та підтримкою окремих додатків для багатьох платформ. По-перше, це впливає на вартість розробки. По-друге, це призводить до значних часових витрат на комунікацію між командами розробки та налагодження відповідності між платформними додатками. Ці фактори призводять до втрати гнучкості розробки та збільшення ціни внесення змін на будь-якому етапі.

Більшість сучасних платформ для крос-платформної розробки використовують системний компонент WebView для створення додатків. Але вони мають свої обмеження та часто приховують реалізацію тих чи інших речей. Наприклад, створення кінцевих збірок користувацьких додатків відбувається на віддаленому сервері і розробник не має доступу до вихідного тексту програми. Підтримують малу кількість цільових платформ, а створення додаткових модулів взаємодії із системними сервісами вимагає специфічних знань цільової платформи.

Відповідно до цього актуальним є створення модифікованого методу роботи з WebView для розроблення мобільних додатків. Що дозволить

легко інтегруватися в існуючі платформи та розробляти користувацькі додатки без специфічних знань цільових платформ.

Отже, об’єктом дослідження в даній роботі є фреймворки для створення крос-платформних користувацьких додатків.

Предметом дослідження є способи інтеграції з "рідними" API мобільних платформ та архітектура.

Таким чином, основною метою цієї роботи є дослідження існуючих рішень для крос-платформної розробки користувацьких додатків, розроблення способу використання WebView для написання крос-платформних користувацьких додатків та розроблення архітектури на основі модульного підходу проектування систем.

Тому відповідно до мети написання дипломного проекту та розроблення програмного додатку поставлені і розв’язані наступні задачі:

- огляд та проведення аналізу існуючих аналогів, зокрема програмних рішень та відомих технологій для виділення їх переваг та недоліків, на основі яких відбувається виявлення вимог для розроблення програми;
- розроблення каналу комунікації та структурування формату повідомлень;
- стандартизування модулів для взаємодії із системними сервісами платформ;
- вибір базової технології для проведення модифікування взаємодії із компонентом WebView;
- реалізація й тестування програмного забезпечення та проведення аналізу отриманого результату.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ РОЗРОБКИ КРОСПЛАТФОМНИХ МОБІЛЬНИХ ДОДАТКІВ

Розробка крос-платформних мобільних додатків передбачає створення вихідного тексту програми, який дає можливість програмам працювати на різних платформах, включаючи iOS, Android та інші. Перевагами такого підходу є швидкість як освоєння технології, так і всього процесу розроблення. Швидкість досягається завдяки можливості повторного використання тексту програми. Підхід «написати один раз, запустити де завгодно» дозволяє розробникам використовувати один текст програми на декількох платформах, що значно зменшує грошові витрати і скорочує час розробки, на відміну від розроблення рідних додатків. Можливість повторного використання тексту програми на декількох платформах дозволяє зменшити затрати ресурсів на розробку до 50-80%.

Затрати часу зменшуються також за рахунок того, що розробникам крос-платформних програм не потрібно вивчати декілька технологій перед створенням своїх додатків. Оскільки немає необхідності у створенні різних баз тексту програми, початкове розгортання на цільових платформах відбувається набагато швидше. Крім того, майбутні зміни в додатку можуть бути здійснені одночасно, не вносячи окремих змін на кожній платформі. Також, не виникає проблеми синхронізації внесення змін для різних платформ.

Така можливість дозволяє швидко та легко охопити більшу кількість платформ та пристроїв, збільшує цільову аудиторію. Але, як і будь-яка технологія, крос-платформний підхід має свої недоліки.

Оскільки кожна платформа має свої API для взаємодії із системними сервісами, написати текст програми, що буде працювати всюди, неможливо. У тексті програми необхідно враховувати на якій платформі у даний момент запущено програму та використовувати різні канали для взаємодії. Тому досить часто для технологій створюється екосистема, що складається з обгортки на кожній платформі та особливого середовища виконання

клієнтського додатку. Така обгортка стає проміжною ланкою, між користувачем додатку та системними API, через що виникають проблеми із продуктивністю роботи додатку та швидкістю відгуку на дії користувача.

Також слід зазначити, що кожна платформа має свої особливі компоненти користувацького інтерфейсу, взаємодія з якими може відрізнятися. Проблемою є те, що забезпечення звичної користувачам поведінки вимагає додаткового часу та робить текст програми платформо-орієнтованим. Це порушує ідею крос-платформного підходу розроблення користувацьких додатків. З іншого боку використання незвичних користувачам компонентів може негативно вплинути на UX [4].

Розроблення крос-платформних додатків може бути реалізовано з використанням різних технологій. Наприклад, мова програмування C, веб-технології, C# (Xamarin) тощо. Для даної роботи було вирішено аналізувати рішення, що використовують веб-технології. Тому, що вони можуть працювати практично на будь-якій платформі: настільних комп'ютерах, телефонах, планшетах, автомобілях, холодильниках тощо. Це досягається за рахунок того, що вони базуються на веб-стандартах та загальних API, які спільно використовуються на цих платформах. Далі у цьому розділі ми розглянемо декілька таких рішень.

1.1. PhoneGap / Cordova

PhoneGap народився в Nitobi Software влітку 2008 року. А у жовтні 2012 року став частиною Apache Software Foundation (ASF) під назвою Apache Cordova. Це дозволило PhoneGap стати назавжди відкритим для інших розробників та мати ліцензію Apache, версії 2.0.

PhoneGap мають два переконання:

1. Проблема крос-платформності може бути вирішена за рахунок веб.
2. Всі технології з часом знецінюються [5].

PhoneGap – це технологія контейнерних програм, яка дозволяє створювати вбудовані програми для мобільних пристроїв.

Інтерфейс користувача для додатків PhoneGap створюється за допомогою HTML, CSS та JavaScript. Шар інтерфейсу програми PhoneGap – це WebView, який займає 100% ширини пристрою та 100% висоти пристрою. Він надає вміст HTML, без прикраси вікна звичайного веб-браузера. Необхідно так збудувати свою програму, щоб скористатися цим простором, та помістити навігаційні / інтерактивні / елементи вмісту програми у свій інтерфейс користувача на базі HTML та CSS.

WebView, який використовується PhoneGap, – це той самий WebView, який використовується в рідній операційній системі. В iOS це клас UIWebView Objective-C; на Android це Android andwebwekit.WebView. Оскільки існують розбіжності в механізмах візуалізації WebView між операційними системами, необхідно переконатися, що це враховано при реалізації інтерфейсу.

PhoneGap надає інтерфейс програмування додатків (API), що дозволяє отримувати доступ до функціональної операційної системи за допомогою JavaScript. Логіка програми будується за допомогою JavaScript, а API PhoneGap обробляє зв'язок із рідною операційною системою. Також до можливостей стандартного API PhoneGap можна додати власні особливі «рідні плагіни» з використанням механізму зв'язку JavaScript-to-Native. Рідні плагіни PhoneGap дозволяють писати власні класи та відповідні інтерфейси JavaScript для використання у ваших додатках.

Програми PhoneGap розробляються за допомогою HTML, CSS та JavaScript, однак кінцевим продуктом програми PhoneGap є двійковий архів додатків, який можна поширювати через стандартні екосистеми додатків. Для додатків iOS вихідний формат файлу IPA (архів додатків для iOS), для додатків Android APK (пакунок Android), для Windows Phone файл XAP (пакет програм) тощо. Це ті самі формати додатків, які використовуються в «рідних» програмах і можуть розповсюджуватися через відповідні екосистеми (iTunes Store, Android Market, Amazon Market, BlackBerry App World, Windows Phone Marketplace тощо).

PhoneGap використовується для написання клієнтських додатків. Зв'язок між клієнтом і сервером додатків може базуватися на стандартних HTTP-запитах для вмісту HTML, служб REST-повної XML, служб JSON або SOAP (або веб-розеток, якщо ваша ОС підтримує це). Це ті самі методи, які ви використовували б у настільному браузері на базі AJAX [6].

Apache Cordova – це проект з відкритою кодовою базою, для розробки мобільних додатків. Він дозволяє використовувати стандартні веб-технології – HTML5, CSS3 і JavaScript для крос-платформної розробки. Програми виконуються в обгортках, орієнтованих на кожну платформу, і покладаються на відповідність стандартам прив'язки API для доступу до можливостей кожного пристрою, таких як датчики, дані, стан мережі тощо.

На рис. 1 зображено архітектуру додатків Cordova.

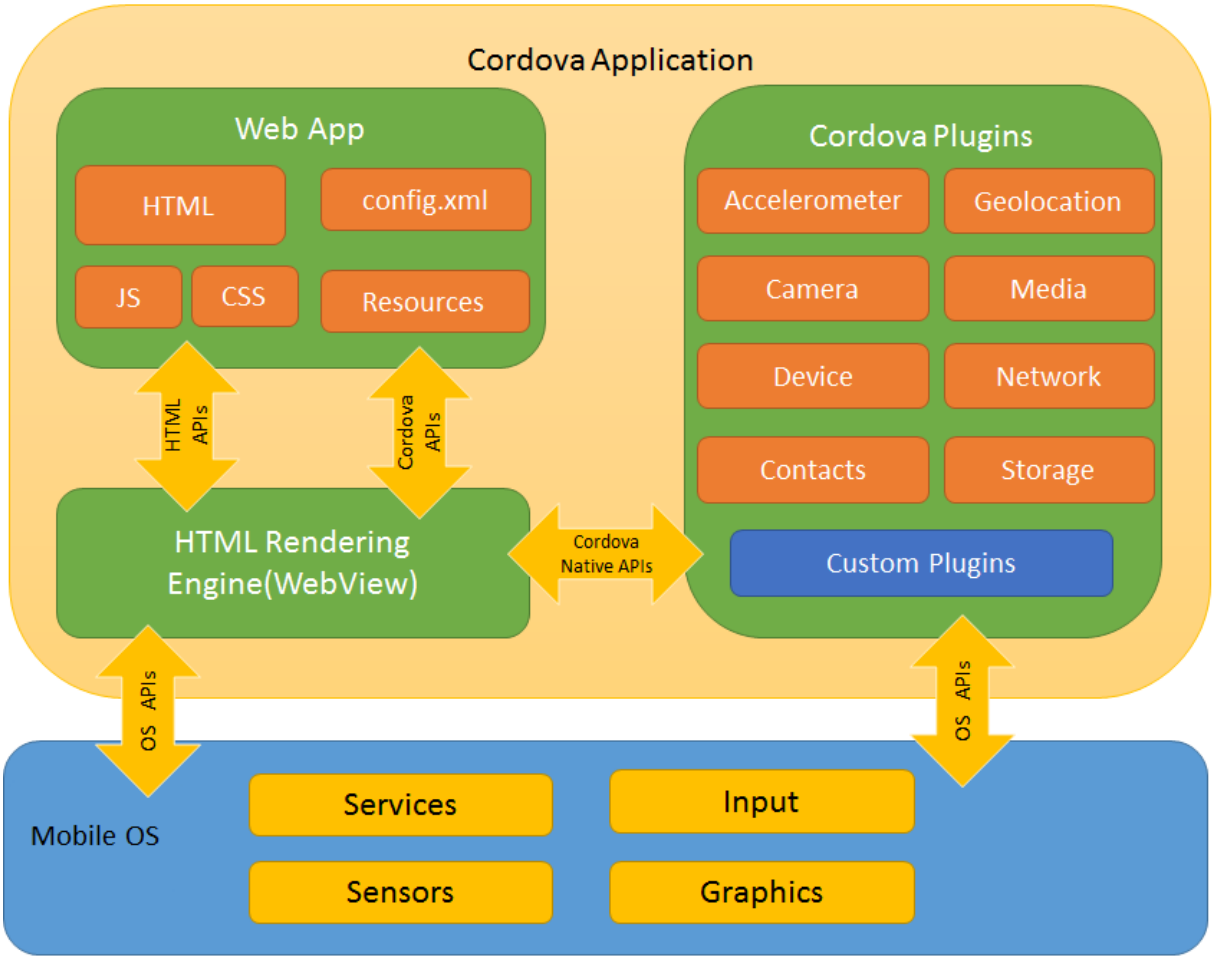


Рис. 1. Архітектура Cordova

WebView. WebView з підтримкою Cordova може надавати програмі весь її інтерфейс користувача. На деяких платформах він також може бути компонентом у більшому, гібридному додатку, який змішує WebView з рідними компонентами додатків.

WebApp. Це частина, де знаходиться ваш додаток. Сама програма реалізована як веб-сторінка, за замовчуванням локальний файл з назвою index.html, на який посилаються CSS, JavaScript, зображення, медіа-файли або інші ресурси, необхідні для його запуску. Додаток виконується у WebView в корінній програмі, яку ви поширюють в магазинах додатків.

Цей контейнер має дуже важливий файл – файл config.xml, який надає інформацію про додаток та вказує параметри, що впливають на його роботу, наприклад, чи відповідає він на зміну орієнтації.

Plugins. Плагіни є невід'ємною частиною екосистеми Cordova. Вони надають інтерфейс для Cordova та "рідних" компонентів для зв'язку між собою та прив'язки до стандартних API пристроїв. Це дає змогу викликати власний текст програми із JavaScript.

Проект Apache Cordova підтримує набір плагінів під назвою Core Plugins. Ці основні плагіни надають вашій програмі доступ до можливостей пристроїв, таких як акумулятор, камера, контакти тощо.

Окрім основних плагінів, є кілька сторонніх плагінів, які забезпечують додаткові прив'язки до функцій, не обов'язково доступних на всіх платформах [7].

1.2. Ionic Framework / Capacitor

Ionic був заснований у 2012 році, коли використання веб-технологій як засобу для створення власних додатків ще було в початковому стані. Основною ідеєю було створити кращий спосіб для веб-розробників використовувати свої наявні набори навичок при створенні додатків для смартфонів.

Ionic Framework – це безкоштовний та відкритий проект, випущений за дозволеною ліцензією MIT. Це означає, що його можна безкоштовно використовувати в особистих або комерційних проектах. MIT – та сама ліцензія, яка використовується в таких популярних проектах, як jQuery і Ruby on Rails.

Ionic Framework – це інструментарій інтерфейсу з відкритим текстом програми для створення мобільних та настільних додатків за допомогою веб-технологій (HTML, CSS та JavaScript).

Ionic Framework орієнтований на інтерфейс користувача або взаємодію із користувальницьким інтерфейсом програми (елементи керування, взаємодії, жести, анімації). Він легко може бути інтегрований з іншими бібліотеками, такими як Angular, або може використовуватися окремо.

На даний час Ionic Framework має офіційні інтеграції з Angular та React, а підтримка Vue знаходиться в розробці [8].

Один з найпоширеніших випадків використання Ionic – це створення додатку, який можна завантажити як з App Store, так і з Play Store. І набори для розробки програмного забезпечення для iOS та Android (SDK) надають WebView, які можуть відобразити будь-який додаток Ionic, і надалі дозволяють отримати повний доступ до SDK.

Такі проекти, як "Capacitor" та "Cordova", зазвичай використовуються для надання додаткам Ionic такого доступу до Native SDK. Це означає, що розробники можуть швидко створити додаток, використовуючи звичайні інструменти веб-розробки, та все ж мати доступ до рідних функцій, таких як акселерометр пристрою, камера, GPS тощо [9].

Capacitor – технологія крос-платформного виконання додатків, що дозволяє створювати веб-додатки, які працюють на iOS, Android, Electron та веб.

Capacitor надає послідовний, орієнтований на набір веб-API, що дозволяє додатку максимально наближатися до веб-стандартів, отримуючи доступ до багатьох функцій рідного пристрою на платформах, які їх підтримують. Додавання «рідного» функціоналу легко за допомогою простого Plugin API для Swift на iOS, Java на Android та JavaScript для Web.

Capacitor – спадкоємець Apache Cordova та Adobe PhoneGap, натхненний іншими популярними інструментами крос-платформної розробки, такими як React Native та Turbolinks, але повністю зосереджений на тому, щоб сучасні веб-додатки могли легко працювати на всіх основних платформах. Capacitor підтримує зворотну підтримку багатьох існуючих плагінів Cordova.

1.3. Flutter

Flutter – це програмний пакет SDK для створення додатків для iOS, Android та Web з однією кодовою базою. Мета полягає в тому, щоб дозволити розробникам розробляти високоефективні програми, які відчувають себе природними на різних платформах. Мовою розробки э Dart.

Як і React Native, Flutter використовує компоненти відображення в реактивному стилі. Однак, якщо React Native перекладається на рідні віджети, то Flutter збирає весь шлях до "рідного" тексту програми. Flutter контролює кожен піксель на екрані, що дозволяє уникнути проблем із продуктивністю, викликаних необхідністю мосту JavaScript.

Dart має такі характеристики:

– забезпечує відкритий текст програми, масштабовану мову програмування для створення веб, серверів та мобільних додатків;
– забезпечує об'єктно-орієнтовану єдину мову успадкування, яка використовує синтаксис стилю C, який AOT-компілюється у рідну мову;
– може бути трансльована в JavaScript;
– підтримує інтерфейси та абстрактні класи.

На iOS більшість створених у користувальницькому інтерфейсі здійснюється за допомогою об’єктів перегляду, що є примірниками класу UIView. Вони можуть діяти як контейнери для інших класів UIView, які формують ваш макет. У Flutter приблизний еквівалент UIView – віджет. Віджети не відображають саме компоненти iOS, але, коли ви знайомитесь із тим, як працює Flutter, ви можете вважати їх «способом оголошення та створення інтерфейсу користувача». Але вони мають декілька відмінностей щодо UIView. Для початку віджети мають різну тривалість життя: вони незмінні та існують лише до тих пір, поки їх не потрібно змінити. Щоразу, коли віджети або їх стан змінюються, рамка Flutter створює нове дерево екземплярів віджетів. Для порівняння, перегляд iOS не відтворюється під час його зміни, а, скоріше, це об'єкт, що змінюється, який малюється один раз і не перемальовується, поки не буде визнаний недійсним за допомогою setNeedsDisplay().

Крім того, на відміну від UIView, віджети Flutter мають легку вагу, частково через їх незмінність. Оскільки вони самі не переглядають і нічого безпосередньо не малюють, а є описом інтерфейсу користувача та його семантики, яка «надувається» у фактичні об’єкти перегляду під кришкою.

Flutter включає в себе бібліотеку Material Components. Це віджети, які реалізують вказівки щодо дизайну матеріалів. Material Design – це гнучка система дизайну, оптимізована для всіх платформ, включаючи iOS [10].

Flutter включає в себе сучасний каркас у стилі реагування, 2D-рендерінг, готові віджети та засоби розробки. Ці компоненти працюють разом, щоб допомогти створювати, тестувати та налагоджувати програми. Все організовано навколо принципу: «Все – віджет».

Віджети – це основні складові користувальницького інтерфейсу програми Flutter. Кожен віджет – це незмінна частина користувальницького інтерфейсу. На відміну від інших фреймворків, що розділяють представлення, контролери перегляду, макети та інші властивості, Flutter має послідовну, єдину модель об'єкта: віджет.

Віджетом може визначатися:

– структурний елемент (наприклад, кнопка або меню);

– стилістичний елемент (наприклад, шрифт або колірна схема);

– аспект планування (як набивання).

Віджети формують ієрархію на основі композиції. Кожен віджет розміщується всередині і успадковує властивості від свого батьківського. Немає окремого об’єкта «додаток». Натомість є кореневий віджет, що виконує цю роль [11].

1.4. React Native

React Native – це фреймворк JavaScript для написання мобільних додатків для iOS та Android. Він базується на React, бібліотеці JavaScript Facebook для створення інтерфейсів користувача, але замість націлення на браузер, він націлений на мобільні платформи. Іншими словами: веб-розробники можуть писати мобільні додатки, які виглядають і відчувають себе справді «рідними» – все в рамках JavaScript бібліотеки (React). Крім того, оскільки більшість написаних вами текстів програм можна поділити між платформами, React Native полегшує одночасну розробку і для Android, і для iOS.

Подібно до React для Інтернету, програми React Native записуються за допомогою суміші JavaScript та XML-розмітки, відомої як JSX. Потім «міст» React Native викликає вбудовані API візуалізації в Objective-C (для iOS) або Java (для Android). Таким чином, додаток відображатиметься за допомогою реальних компонентів мобільного інтерфейсу, а не веб-перегляду, і буде виглядати як будь-який інший мобільний додаток. React Native також відкриває інтерфейси JavaScript для API платформ, тому ваші додатки React Native можуть отримувати доступ до функцій платформи, таких як камера телефону чи місцезнаходження користувача.

На даний момент React Native підтримує iOS та Android, та він має можливість розширитись і на майбутніх платформах. Переважна більшість

тексту програми, написаного на React Native, буде крос-платформною. Такі компанії, як: Facebook, Palantir і TaskRabbit – вже використовують його у розробленні програм, орієнтованих на користувачів.

Той факт, що React Native фактично відображає за допомогою стандартних API візуалізації своєї хост-платформи, дозволяє йому виділятися з більшості існуючих методів розвитку платформних додатків, таких як Cordova або Ionic. Існуючі методи написання мобільних додатків за допомогою комбінацій JavaScript, HTML та CSS зазвичай відображаються за допомогою веб-перегляду. Хоча такий підхід може працювати, він також має недоліки, особливо щодо ефективності. Крім того, вони зазвичай не мають доступу до "рідних" елементів інтерфейсу хост-платформи. Коли ці рамки намагаються імітувати природні елементи інтерфейсу, результати зазвичай «відчуваються» лише трохи; реверсивна інженерія всіх тонких речей, таких як анімація, вимагає величезних зусиль, і вони можуть швидко застаріти.

На відміну від цього, React Native фактично переводить вашу розмітку на реальні, рідні елементи інтерфейсу, використовуючи існуючі засоби візуалізації поглядів на будь-якій платформі, з якою ви працюєте. Крім того, React працює окремо від основного потоку інтерфейсу, тому ваша програма може підтримувати високу продуктивність без шкоди для можливості. Цикл оновлення в React Native такий же, як і в React: коли параметри або стан змінюються, React Native повторно перемальовує інтерфейс. Основна відмінність React Native від React у веб-переглядачі полягає в тому, що React Native це робить, використовуючи бібліотеки інтерфейсу користувача на своїй хост-платформі, а не використовуючи розмітку HTML та CSS.

Для розробників, які звикли працювати в Інтернеті з React, це означає, що вони можуть писати мобільні додатки з продуктивністю та виглядом "рідного" додатка, використовуючи при цьому звичні інструменти. React Native також означає покращення порівняно з нормальним мобільним

розвитком у двох інших сферах: досвід розробника та потенціал розвитку платформ [1].

Робота з React Native може різко скоротити ресурси, необхідні для створення мобільних додатків. Будь-який розробник, який вміє писати текст програми з використанням React, тепер може орієнтуватися на Інтернет, iOS та Android, і все з тим самим набором навичок. Видаляючи необхідність синхронізувати розробників на різних цільових платформах, React Native дозволяє вашій команді швидше вносити зміни та ефективніше ділитися знаннями та ресурсами.

Окрім спільних знань, також можна поділитися великою частиною тексту програми. Не весь текст програми, який пишеться, буде крос-платформним, тому залежно від того, яка функціональність потрібна на певній платформі, розробникам може час від часу знадобиться занурюватися в Objective-C або Java. Але повторно використовувати текст програми на платформах напрочуд легко з React Native. Наприклад, додаток Facebook Ads Manager для Android поділяє 87% своєї кодової бази з версією iOS, як зазначено в програмі React Europe 2015.

Як і в усьому, використання React Native не позбавлене й мінусів, і те, чи дійсно використання React Native як технології буде ефективнішим, залежить від індивідуальної ситуації.

Найбільший ризик – це, ймовірно, зрілість React Native, оскільки проект ще відносно молодий. Підтримка iOS була випущена в березні 2015 року, а підтримка Android була випущена у вересні 2015 року. Документація, безумовно, має можливість вдосконалитись і продовжує розвиватися. Деякі функції на iOS та Android все ще не підтримуються, і спільнота все ще шукає найкращі практики. Перевагою є те, що в переважній більшості випадків доступна можливість самостійно реалізувати підтримку відсутніх API.

1.5. Висновки до розділу 1

У даному розділі було проведено аналіз існуючих рішень для крос-платформної розробки. Отримана інформація була систематизована з метою використання в подальшій роботі для визначення вимог до програмної розробки, що дозволить створювати крос-платформні користувацькі додатки.

Перевагою більшості рішень є повторне використання тексту програми для виконання програми на різних платформах. Також, в якості однієї з переваг можна виділити мову розробки – JavaScript, оскільки вона є простою для вивчення та найпопулярнішою серед розробників станом на жовтень 2019 року [12]. Недоліками більшості розглянутих рішень є необхідність працювати всередині екосистеми та важко розширюваної обгортки. Більш детальні результати за критеріями порівняння наведено у табл. 1.

Таблиця 1

Порівняння існуючих рішень

Критерій	PhoneGap	Ionic	Flutter	React Native
Рік релізу платформи	2008	2012	2017	2015
Мова програмування	JavaScript	JavaScript	Dart	JavaScript
Спільнота (розробники на GitHub)	3914	341	477	2038
Додаткова обгортка	Cordova	Capacitor / Cordova	–	–
Платформи, що підтримуються	iOS, Android, Windows, Electron, Web, Mac OS X	iOS, Android, Web/PWA, Electron	iOS, Android, Web	iOS, Android
Середовище виконання	WebView	WebView		JS Thread (JS Core)

Продовження табл. 1

Можливість повторного використання тексту програми	На всіх платформах	На всіх платформах	На всіх платформах	Наявний специфічний для платформи текст програми
Рівень покриття "рідного" API плагінами	Середній	Низький	Низький	Високий
Складність написання "рідних" плагінів	Складно	Середнє	Середнє	Легко

На основі дослідження крос-платформної розробки та порівняння існуючих рішень за критеріями було сформовано мету та вимоги до розроблюваного в даній роботі програмного продукту.

Метою даної роботи є розроблення способу використання WebView для написання крос-платформних користувацьких додатків та розроблення.

Вимогами є:

– розроблення базової обгортки, що має виступати транслятором: для тексту програми клієнтського додатку надає єдиний API для взаємодії з системними сервісами і використовує необхідні команди API залежно від платформи, на якій зараз виконується програма;
– розроблення архітектури для взаємодії з "рідними" API;
– програма має виконуватися як на мобільних пристроях Android, iOS, так і у веб-переглядачі;
– в якості базової технології для мобільних платформ має виступати React Native. Оскільки дана технологія має найактивнішу спільноту, найширше покриття "рідного" API, використовує JavaScript в якості мови програмування та має найпростіший інтерфейс для написання додаткових "рідних" плагінів.

Виклад рішення наведено у розділі 2 цієї дисертації.

2. МОДИФІКОВАЦІЯ ПІДХОДУ ДО ВИКОРИСТАННЯ WEBVIEW ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

Рішення, що пропонується в даній дисертації, сформувалося в результаті поступового вирішення проблем. Тому даний розділ описуватиме проблеми, задачі та їх вирішення у тому порядку, як вони виникали в процесі роботи.

Виходячи з того , що основною метою цієї роботи є розробка такої платформи, за допомогою якої буде зручно створювати користувацькі додатки мовою програмування JavaScript, та надати можливість використовувати їх як у веб-браузері, так і на мобільних платформах у вигляді "рідних" додатків з використанням "рідного" API операційних систем смартфонів.

2.1. Виконання тексту програми в окремому потоці

JavaScript не підтримує багатопотоковість, оскільки інтерпретатор мови у браузері є однопотоковим. Також, сучасні браузери не дозволять одночасно запускати декілька JavaScript-потоків на одній веб-сторінці, оскільки це може спричинити великі проблеми конкурування у вже існуючих веб-сторінках.

React Native використовує той самий інтерпретатор, що і браузер, для відтворення тексту програми на JavaScript у мобільних додатках. Тому такі мобільні додатки мають ту саму проблему – однопотоковість.

Проблема полягає в тому, що у додатку один і той самий потік відповідає за реакцію на взаємодію користувача з інтерфейсом та виконання обчислень. Якщо ж обчислення виконуються довго, додаток не може реагувати на дії користувача. Тому створюється враження неспроможності додатка працювати і користувач, найімовірніше, просто його закриває, не дочекавшись завершення виконання обчислень.

Однак, існує спосіб створити емуляцію паралельності за допомогою використання *setTimeout*, щоб виконати певні обчислення. Використання

setTimeout призводить до того, що додаток починає виконувати текст програми після заданої кількості мілісекунд і не втрачає контроль над потоком візуалізації моментально. Тим самим дозволяє обробити початок ресурсномістких обчислень та відобразити відповідний показник процесу. Тим самим не створювати враження неспроможності додатка працювати.

Оскільки, додатки, написані за допомогою React Native, – це "рідні" додатки для різних операційних систем вони мають більше можливостей, ніж браузер. Однією з таких можливостей є використання вбудованого у додаток веб-переглядача – WebView.

Клас WebView – це розширення класу View, що дозволяє відображати веб-сторінки як частину макета вашої сторінки. Він має обмеження, порівняно із повноцінним веб-браузером. Наприклад, не має таких компонент, як елементи навігації або адресний рядок. Все, що робить WebView, за замовчуванням – це відображає веб-сторінку.

Загальний сценарій використання WebView – це відображення інформації у додатку, яку, можливо, потрібно буде оновити, наприклад, угода з кінцевим користувачем чи посібник користувача. У мобільних додатках можна створити сторінку, що містить WebView, а потім використовувати її для відображення документа, який розміщено в Інтернеті.

Ще один сценарій, в якому WebView може допомогти, це якщо ваш додаток надає користувачеві дані, які завжди потребують підключення до Інтернету. Наприклад, дані електронної пошти. У цьому випадку простіше створити WebView у "рідному" додатку, який показує веб-сторінку з усіма даними, а не виконує мережевий запит, потім аналізує дані та відображає їх [13].

WebView має можливість відображати вміст веб-сторінок не тільки за посиланням, а й зчитувати його із рядка. Це дає змогу завантажити та відтворити будь-який фрагмент тексту програми на мові JavaScript у WebView із середини додатка.

Розглянувши організацію потоків у мобільних додатках, написаних з використанням React Native, можна визначити 4 потоки, що виділяються для додатку. На рис. 2 показано розподіл потоків між "рідною" та JavaScript Core частинами додатка.

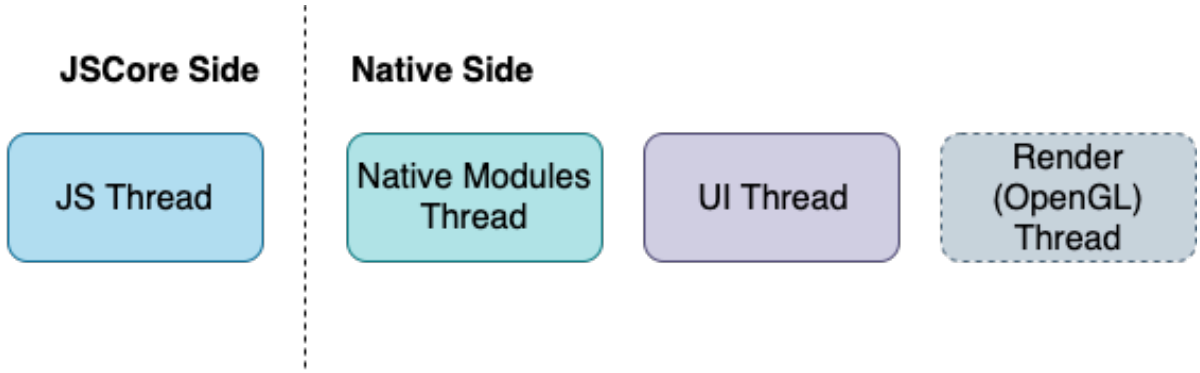


Рис. 2. Потоки у React Native

UI Thread, також відомий як *Main Thread,* використовується для відображення компонент "рідного" інтерфейсу Android та iOS.

JS Thread – це потік, де буде працювати логіка. Це потік, в якому виконується текст програми на мові JavaScript, здійснюються виклики API, обробляються події дотику та багато інших. Оновлення, що стосуються "рідних" компонентів інтерфейсу збираються та надсилаються на "рідну" сторону в кінці кожного циклу подій в *JS Thread* (і виконуються зрештою в *UI Thread*).

Native Modules Thread – потік "рідних" модулів. Використовується, коли додатку потрібен доступ до API платформи.

Render Thread – потік відображення. Присутній тільки в Android L (5.0) та використовується для створення фактичних команд до OpenGL для відображення інтерфейсу користувача [14].

Також, коли додаток викликає веб-переглядач у себе всередині, для переглядача операційна система смартфону виділяє ще один окремий потік для виконання та, як показано на рис. 3, розміщує його на стороні "рідного" додатку.

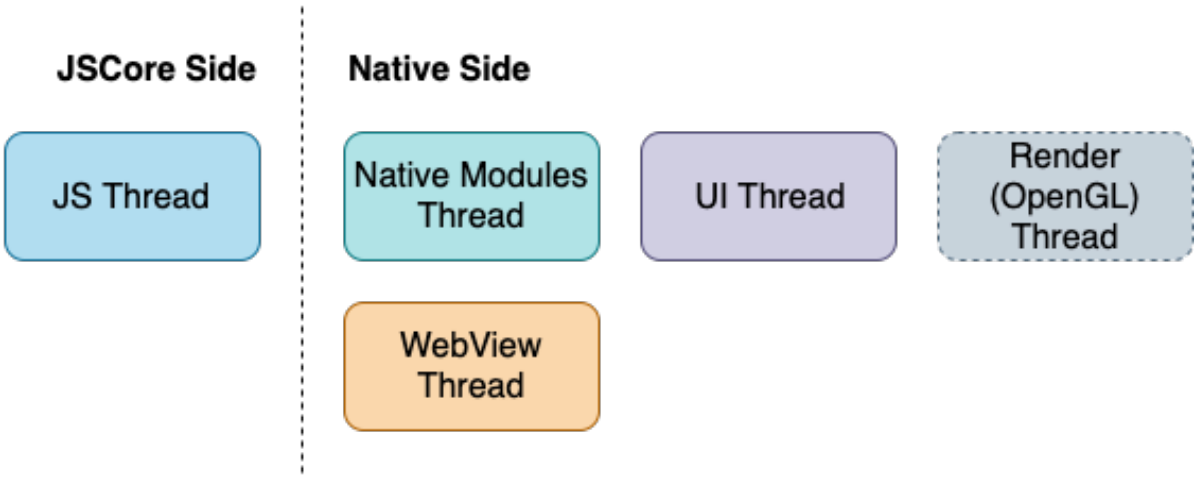


Рис. 3. Потоки у React Native з WebView

Отже, WebView можна використовувати, як середовище для виконання ресурсномістких операцій та розрахунків, оскільки він запускається в окремому потоці операційної системи. Таким чином, користувацький інтерфейс не буде блокуватися. Текст програми на мові JavaScript має бути переданий до WebView у вигляді рядка, а сам компонент має бути невидимим для користувача. Приклад коду наведено у лістингу 1.

Лістинг 1. Використання компонента WebView

```
import React from 'react';
import { WebView } from 'react-native-webview';

const jsString = require('./dist/jsString.js');

export default class WebViewCode extends React.Component {
    render() {
        return (
            <WebView
                source = {{ html: '<div id="message"></div>' }}
                style = {{ width: 0, height: 0, position:
'absolute' }}
                injectedJavaScript = {jsString}
                javaScriptEnabled = {true}
                javaScriptEnabledAndroid = {true}
            />
        );
    }
}
```

2.2. Канал комунікації

Оскільки на попередньому етапі було прийнято рішення використовувати WebView в якості середовища для виконання ресурсномістких операцій, виникла необхідність у комунікації між двома потоками виконання програми.

Єдиним варіантом, що дозволяють реалізувати API WebView для React Native та API для браузера, є використання інтерфейсу MessageEvent для обміну повідомленнями між документами.

Для WebView – це такі складові API, як метод *postMessage* та делегат *onmessage*, що доступні із тексту програми на мові JavaScript з глобального об'єкту *window*.

Window.postMessage – метод, що дозволяє безпечно відправляти кросдоменні запити. Зазвичай сценаріям на різних сторінках дозволений доступ один до одного тільки якщо сторінки, які їх виконували, передаються по одному протоколу (зазвичай це https), номер порту (443 – за замовчуванням для https) і хост (modulo Document.domain встановлений сторінками на одне і теж значення). *Window.postMessage* надає контрольований механізм, щоб обійти це обмеження способом, який безпечний при правильному використанні.

При виклику методу *window.postMessage* браузер викликає MessageEvent для відправки в цільовому вікні, коли завершується будь-який сценарій, який повинен бути виконаний (наприклад, залишилися обробники подій, якщо *window.postMessage* викликати з обробника подій раніше заданих, що очікують таймаутів). MessageEvent має тип message. Властивість *data*, значення якого встановлює перший аргумент в методі *window.postMessage*. Властивість *origin* відповідає адресі цільового документа. Властивість *source* вказує на вікно, з якого *window .postMessage* викликали [15].

Зі сторони React Native є доступною бібліотека із "рідним" компонентом WebView, який також має метод *postMessage* і обробник подій *onMessage* у своєму API.

Модель обміну повідомленнями за допомого *postMessage* та *onMessage* можна віднести до асинхронної подіє-керованої архітектури. У розрізі нашої задачі, це означає, що React Native і WebView можуть відправляти повідомлення один одному без можливості дочекатися відповіді. Це є проблемою, оскільки коли починається виконання розрахунків, необхідно якимось чином отримати результати.

Для вирішення цієї проблеми було створено обгортку, що повертає promise на будь-який запит та закриває його, коли отримує відповідь від іншої сторони.

Також, для досягнення поставленої задачі було вирішено використовувати протокол JSON-RPC версії 2.0.

JSON-RPC – це легкий протокол дистанційного виклику процедури, що не залежить від стану системи. В першу чергу ця специфікація визначає кілька структур даних та правила їх обробки. Транспортний агностицизм полягає в тому, що ця концепція може використовуватися в межах одного процесу через сокети, через HTTP або в багатьох різних середовищах передачі повідомлень. У якості формату даних використовується JSON (RFC 4627).

Оскільки JSON-RPC використовує JSON, він має таку ж систему типів. JSON може представляти чотири примітивні типи (рядки, числа, булеві значення і NULL) та два структуровані типи (об’єкти та масиви). Усі імена учасників, що обмінюються між Клієнтом та Сервером, які вважаються відповідними будь-якого виду, слід вважати чутливими до регістру. Терміни функція, метод та процедура можна вважати взаємозамінними.

Клієнт визначається як генератор об'єктів "Запит" та обробник об'єктів "Відповідь". Сервер визначається як генератор об'єктів "Запит" та

обробник об'єктів "Відповідь". Одна реалізація цієї специфікації може легко заповнити обидві ці ролі, навіть в той же час, іншим клієнтам або тому ж клієнту [16].

Розглянемо структуру об’єкта "Запит". Він має такі поля:

– jsonrpc – рядок із зазначенням версії протоколу JSON-RPC. Має бути "2.0".
– method – рядок, що містить ім'я методу, який слід викликати. Назви методів, які починаються зі слова rpc, за яким слідує символ періоду (U+002E або ASCII 46), зарезервовані для внутрішніх методів і розширень rpc, і не повинні використовуватись ні для чого іншого.
– params – Структуроване значення, яке містить значення параметрів, які будуть використані під час виклику методу. Це поле може бути пропущене.
– id – ідентифікатор, встановлений Клієнтом, який обов’язково містить значення рядок, число або NULL. Якщо цього поля немає, передбачається, що це сповіщення. Якщо це поле приймає значення – число, то воно не має містити дробових частин. Сервер повинен відповісти з тим самим значенням цього поля в об'єкті "Відповідь". Це поле використовується для співвіднесення контексту між двома об'єктами.

Коли здійснюється rpc-запит, Сервер зобов’язаний відповідати, за винятком випадків, коли це стосується сповіщень. Відповідь виражається як один об’єкт JSON з такими полями:

– jsonrpc – рядок із зазначенням версії протоколу JSON-RPC. Має бути "2.0".
– result – значення цього поля визначається методом, викликаним на сервері. Це поле обов’язково має бути присутнє при успішному виконанні запиту та не має існувати, якщо сталася помилка.
– error – значення для цього поля має бути Об'єктом, структура якого визначена нижче. Це поле є обов’язковим при виникненні помилки

під час виконання запиту. Та не має існувати при успішному виконанні запиту.

– id – це поле є обов’язковим. Воно має містити те саме значення, що і поле відповідного об’єкту "Запит". Якщо при виявленні ідентифікатора в об’єкті "Запит" сталася помилка (наприклад, помилка розбору / недійсний запит), це поле обов’язково бути NULL.

Коли rpc-запит стикається з помилкою, Об'єкт відповіді обов’язково має містити поле *error* зі значенням, яке є об'єктом з такими полями:

– code – число, яке вказує на тип помилки, що сталася. Це поле має бути цілим числом.

– message – рядок із коротким описом помилки.

– data – примітивне або структуроване значення, яке містить додаткову інформацію про помилку. Це поле може бути пропущеним. Значення цього поля визначається сервером.

JSON-RPC має зарезервовані коди помилок для визначених помилок за замовчуванням. Будь-який код у діапазоні від –32768 до –32000, що не визначений у табл.2, зарезервований для подальшого використання. Коди помилок майже такі самі, як запропоновані для XML-RPC [16].

Таблиця 2

Зарезервовані помилки у JSON-RPC

Код	Повідомлення	Опис
–32700	Parse error	Недійсний JSON був отриманий сервером. Під час розбору тексту JSON на сервері сталася помилка.
–32600	Invalid Request	Надісланий JSON не є дійсним об'єктом запиту.
–32601	Method not found	Метод не існує / недоступний.
–32602	Invalid params	Неприпустимі параметри методу.
–32603	Internal error	Внутрішня помилка JSON-RPC.

Продовження табл. 2

від –32000 до 32099	Server error	Зарезервовано для помилок, визначених сервером.

Отже, у розрізі задач цієї дисертації та на даному етапі створення текст програми, що знаходиться на стороні WebView є Сервером і має такий вигляд:

Лістинг 2. Текст програми сервера на стороні WebView

```
import calculate from './calculate';

async function receiveMessage(event) {
    const request = event.data;
    const response = { id: null };

    if (!request) {
        response.error = { code: -32700, message: 'Parse error' };
    } else {
        const { method, id, params } = JSON.parse(request);

        response.id = id;

        if (!calculate[method]) {
            response.error = { code: -32601, message: 'Method not
found' };
        } else {
            try {
                const result = await calculate[method](params);

                response.result = result;
            } catch (error) {
                response.error = { ...error, code: -32000 };
            }
        }

    }

    ReactNative.postMessage(response);
}

window.addEventListener("message", receiveMessage, false);
```

Де поле запиту *method* є назвою однієї з процедур обчислень, що мають виконуватися в окремому потоці на стороні WebView, а поле *params* містить параметри для цієї процедури.

Текст програми на стороні React Native виконує роль Клієнта.

Лістинг 3. Текст програми Клієнта на стороні React Native

```
import React from 'react';
import WebViewCode from './WebViewCode.js';

export default class App extends Component {
    constructor(...args) {
        super(...args);
        this.webViewRef = React.createRef();
    }
    runCalculation = async () => {
        try {
            const result = await
this.webViewRef.current.sendRequest({
                method: 'multiply',
                params: [ 3, 6 ]
            });;

            console.log({ result });
        } catch (error) {
            console.error(error);
        }
    }
    render() {
        return (
            <View>
                <Button onClick={this.runCalculation}>Run
Calculation</Button>
                <WebViewCode ref={this.webViewRef}/>
            </View>
        );
    }
}
```

Також, текст програми на стороні React Native має обгортку, яка формує запити від Клієнта та співвідносить відповіді від Сервера із ними.

Лістинг 4. Текст програми обгортки на стороні React Native

```
import React from 'react';
import { WebView } from 'react-native-webview';

const jsString = require('./dist/external-lib.js');

export default class WebViewCode extends React.Component {
    constructor(...args) {
        super(...args);
        this.resolvers = {};
    }
    sendRequest = (data) => {
        if (this.webView) {
            const id = Date.now();
            this.webView.postMessage(JSON.stringify({ ...data, id
}));

            return new Promise(resolve => this.resolvers[id] =
resolve);
        }
    }
```

```
    _receiver = (event) => {
        const { id, type, ...data } =
JSON.parse(event.nativeEvent.data);

        if (this.resolvers) {
            if (this.resolvers[id]) this.resolvers[id](data);
            delete this.resolvers[id];
        }
    }
    render() {
        return (
            <WebView
                ref = {webView => this.webView = webView}
                onMessage = {this._receiver}
                source = {{ html: '<div id="message"></div>' }}
                style = {{ width: 0, height: 0, position:
'absolute' }}
                injectedJavaScript = {jsString}
                javaScriptEnabled = {true}
                javaScriptEnabledAndroid = {true}
            />
        );
    }
}
```

2.3. Крос-платформеність додатку

Зважаючи на те, що спочатку задача була оптимізувати ресурсномісткі обчислення, підхід, що описано у попередніх двох підрозділах цього розділу, можна вважати остаточним вирішенням проблеми. Але у процесі роботи виникла наступна задача: зробити так, щоб один і той самий текст програми можна було виконувати у трьох середовищах: iOS, Android та web. Причому, було необхідним зберегти можливість клієнтського додатку, запущеного на мобільній платформі, взаємодіяти із "рідними" API операційних систем смартфонів.

Відповідно, маючи результати попередньої частини дослідження, було прийнято рішення відобразити користувацький інтерфейс на стороні WebView. Тому що, середовище WebView дуже подібне до середовища виконання тексту програми на мові JavaScript у веб-браузері. А через канал комунікації між WebView та React Native доступатися до "рідними" API операційних систем.

Таким чином було досягнуто можливість виконання тексту програми на трьох платформах і збережено можливість доступатися до "рідних" API.

Але це призвело до логічних змін в архітектурному сенсі. Тепер роль Сервера взяв на себе текст програми на стороні React Native, а роль Клієнта – на стороні WebView. На рис. 4 схематично зображено архітектуру створюваної екосистеми та потік запиту для отримання географічних координат мобільного пристрою.

Оскільки ролі змінилися на протилежні, то обгортка для синхронізації запитів і відповідей також змістилася на сторону Клієнта.

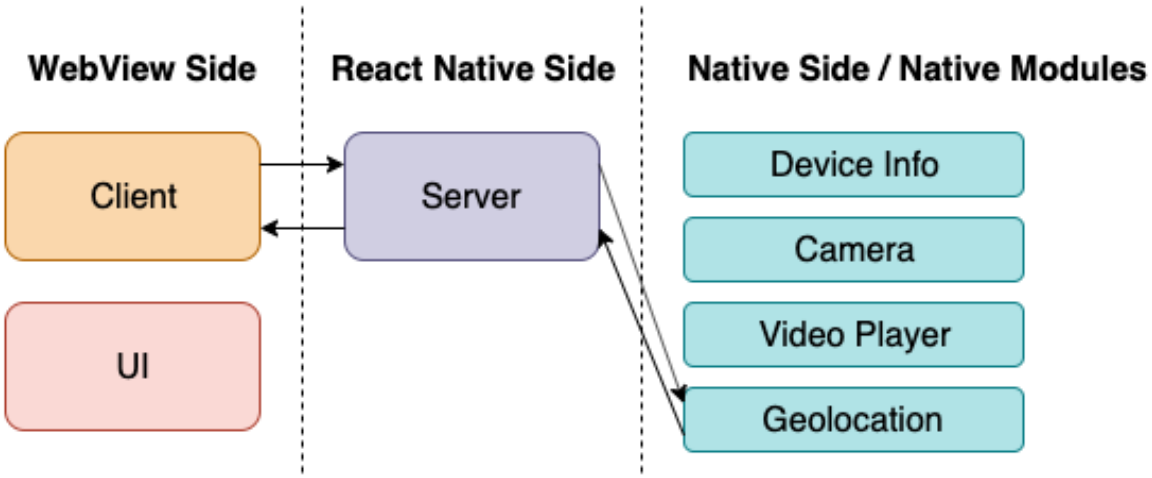


Рис. 4. Схема доступу до "рідного" API

2.4. Двонаправлений канал комунікації

Дана частина стосується організації двонаправленої комунікації між WebView та React Native частинами. Таке рішення було прийнято через необхідність на стороні Клієнта (WebView) отримувати сповіщення про зміну стану додатку (active, inactive, background) та про зміни стану з'єднання з інтернетом для обробки фонового режиму роботи додатка і офлайн режиму.

Тому було продубльовано логіку Сервера та Клієнта на обох сторонах додатка. І архітектура додатка прийняла такий вигляд, як показано на рис.5.

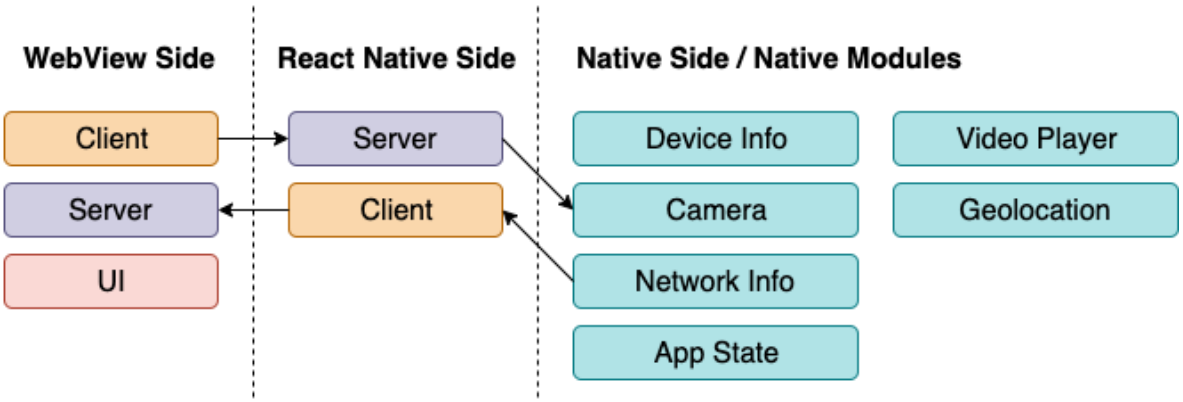


Рис. 5. Архітектура додатка (версія 1)

Для реалізації двонаправленого каналу комунікації було обрано бібліотеку mole-rpc, що базується на JSON-RPC версії 2 і надає інтерфейс для екземплярів Сервера та Клієнта незалежно від транспорту, що використовується для комунікації.

Mole-RPC має такі характеристики:

– транспортний агностицизм – може інтегруватися із будь-яким протоколом передачі даних: HTTP, MQTT, WebSocket, Browser postMesssage;

– не має залежностей від сторонніх бібліотек;

– підтримує двонаправлені веб-сокети через WS-транспорт: сервер, який обробляє віддалені виклики процедури, може відправляти команди у зворотному порядку, використовуючи те саме з'єднання. Отже, ви можете використовувати з'єднання, ініційоване будь-якою з сторін для сервера та клієнта в один і той самий час;

– сервер може використовувати декілька транспортів одночасно;

– підтримка всіх функцій JSON-RPC 2.0: сповіщення, збирання повідомлень у партії;

– легкість створення власного транспорту комунікації [17].

З рішенням проведення інтеграції із mole-rpc, попередню реалізацію Сервера та Клієнта було розділено відносно API, що вимагає дана бібліотека і написано власний транспорт, що базується на API браузера

(*window.postMessage, window.onMessage*) та аналогічному API WebView – React Native компоненти.

Клас транспорту було спроектовано строго за вимогами бібліотеки, а "генераторами" повідомлень стали такі структурні компоненти, як *WindowListener* та *WebViewListener*. Це, відповідно, клас на стороні браузера та компонент на стороні React Native, посилання на екземпляри яких реєструються у класі транспорту. Архітектуру каналу комунікації додатка зображено на рис. 6.

Отже, після введення двонаправленого каналу комункації, додаток отримав можливість обмінюватися повідомленнями між сторонами WebView та React Native у двох напрямах. За рахунок цього стало можливим створити на стороні WebView обробників подій від "рідних" модулів. Також, після інтеграції із бібліотекою mole-rpc, нам вдалось уникнути дублювання тексту програми за рахунок використання API Сервера та Клієнта з бібліотеки. Тому задача звелась до створення транспорту.

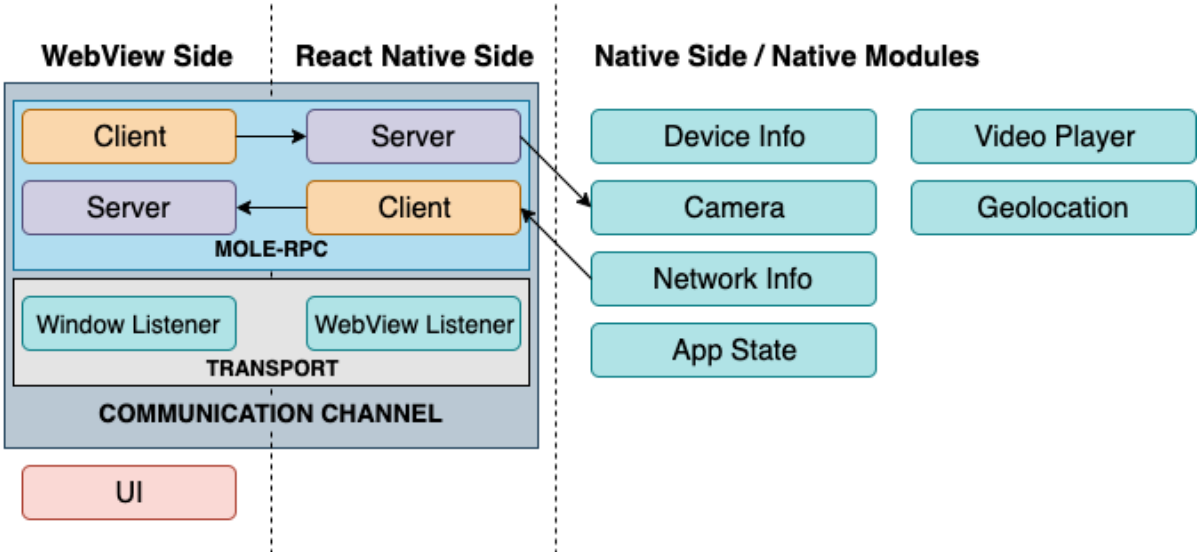


Рис. 6. Архітектура каналу комунікації додатка

2.5. Організація платформо-залежного тексту програми

Коли створюються крос-платформні додатки, необхідно врахувати кожну платформу, на якій виконуватиметься текст програми та описати

взаємодію із конкретними API, що можуть і найчастіше різняться для різних платформ. Це призводить до роздування кодової бази через наявність великої кількості платформо-залежного тексту програми, що використовується лише однією з платформ.

На жаль, зменшити кількість такого тексту програми не є можливим, тому було прийнято заходи щодо його організації. Основною задачею є надання однакового інтерфейсу для взаємодії з API різних платформ. За основу було взято підхід, що використовується у React Native.

React Native дозволяє виконувати виклики функцій між мовами. Тобто ви можете викликати виконання "рідного" тексту програми із JS-сторони та навпаки. Залежно від сторони це досягається різними способами. На стороні "рідного" тексту програми використовується механізм подій для планування виконання функції обробника в JS. Тоді як на стороні React Native безпосередньо викликаються методи, експортовані "рідними" модулями. Використання подій не дає гарантій щодо часу виконання, оскільки подія обробляється окремим потоком.

Події є потужними, оскільки вони дозволяють змінювати компоненти React Native, не потребуючи посилання на них. Однак є деякі вузькі місця, в які можна потрапити, використовуючи їх:

– Оскільки події можна надсилати з будь-якого місця, вони можуть вносити залежності у текст програми в стилі спагеті.
– Події поділяють простір імен, що означає, що ви можете зіткнутися з колізіями імен. Такі колізії не можуть бути виявлені статично, що ускладнює їх налагодження.
– Якщо використовується кілька екземплярів одного компонента React Native і є необхідність відрізнити їх з точки зору події, то, ймовірно, потрібно буде вводити ідентифікатори та передавати їх разом із подіями.

Загальна модель, яка використовується під час вбудовування "рідного" в React Native, полягає в тому, щоб зробити RCTViewManager

"рідного" компонента делегатом для переглядів, пересилаючи події назад в JavaScript через міст. Це зберігає споріднені виклики подій в одному місці.

"Рідні" модулі – це класи Objective-C або Java, які доступні в JavaScript. Зазвичай створюється один екземпляр кожного модуля для одного JavaScript-мосту.

Той факт, що "рідні" модулі присутні в одному екземплярі, обмежує механізм у контексті вбудовування. Скажімо, у нас є компонент React Native, вбудований у власний вигляд, і ми хочемо оновити "рідне", батьківське представлення. Використовуючи механізм "рідного" модуля, ми експортуємо функцію, яка приймає не тільки очікувані аргументи, але і ідентифікатор батьківського власного виду. Ідентифікатор буде використаний для отримання посилання на батьківський вигляд для оновлення. З огляду на це, нам потрібно буде зберегти відображення від ідентифікаторів до "рідних" представлень у модулі [18].

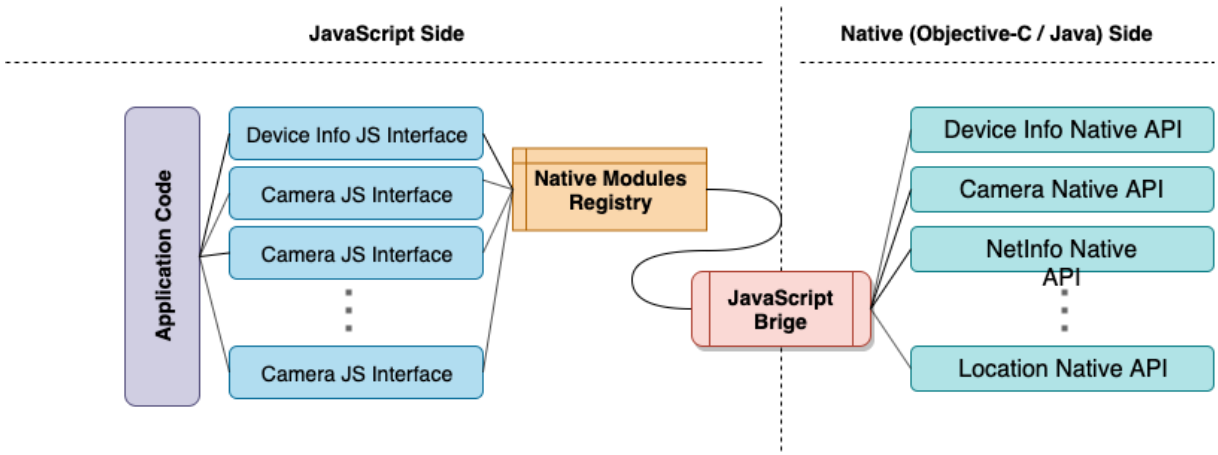


Рис. 7. Схема організації крос-платформного тексту програми на React Native

На рис. 7 та рис. 8 схематично зображено організацію "рідних" модулів та взаємодію між ними та текстом програми додатка у React Native та у розроблюваній системі. Для React Native можна виділити такі основні складові, як:

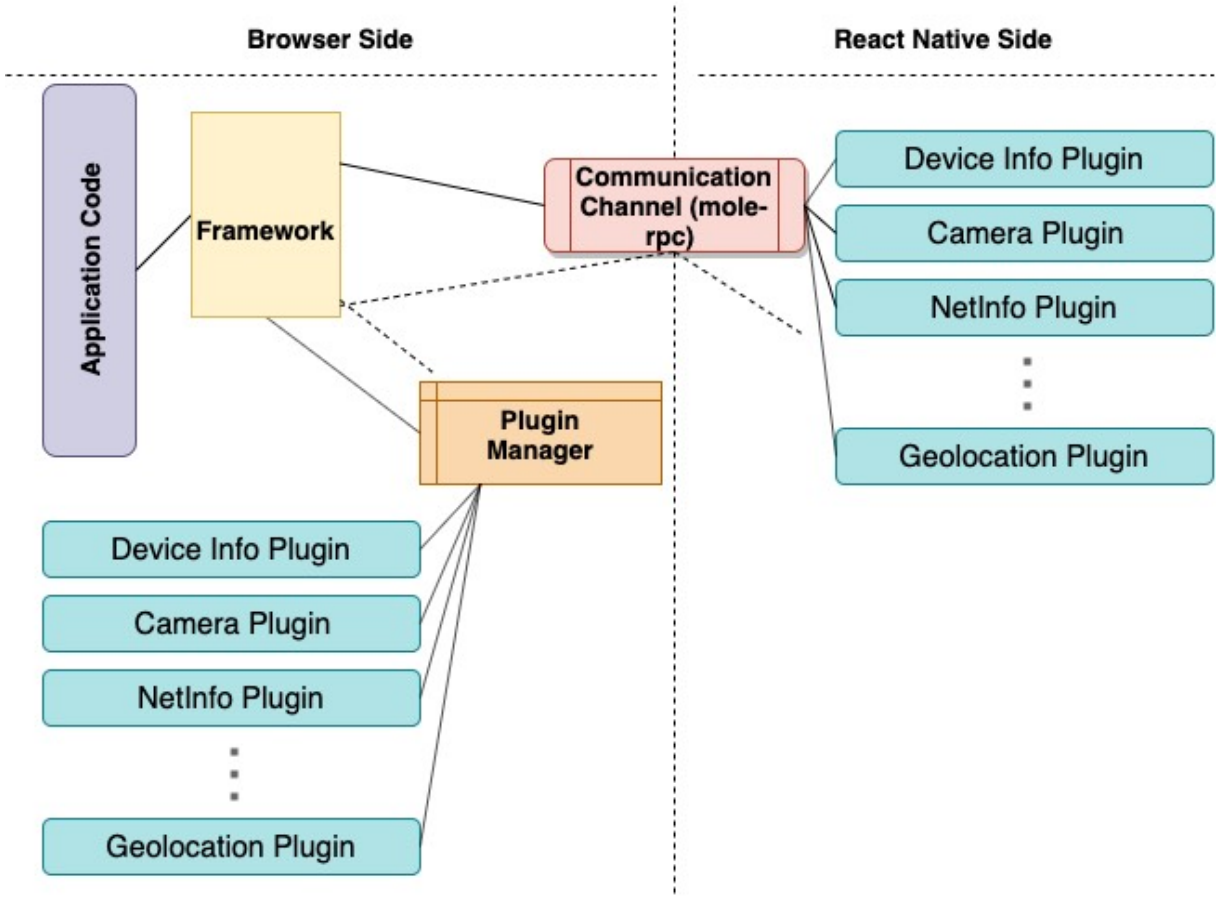


Рис. 8. Схема організації крос-платформного тексту програми у розроблюваній системі

– JavaScript Bridge – канал, через який відбувається обмін повідомленнями між сторонами. У нашій системі цю роль виконує канал комунікації, який використовує бібліотеку mole-rpc та працює за протоколом віддаленого виклику процедури (RPC). Цей канал детально описаний у підрозділах 2.4 та 3.2 цієї дисертації.
– Native API – інтерфейси, що знаходяться у "рідному" тексті програми (Objective-C або Java) для доступу до системних API платформи та які реєструють методи у Native Modules Registry для надання можливості їх викликати з тексту програми користувацького додатка. У розроблюваній системі те ж саме виконують «Плагіни».
– Native Modules Registry – глобальний об’єкт, що містить у собі інформацію про всі Native API, доступний у тексті програми

користувацького додатка. Він з’єднує запит від користувацького додатка на доступ до Native API та відповідь. У розроблюваній системі цю роль виконує «Менеджер Плагінів».

– JS Interface – це класи / компоненти, що доступні із тексту програми користувацького додатка. Вони знають до якого конкретно "рідного" модуля у Native Modules Registry необхідно виконати запит. Вони є публічними інтерфейсами для зручності виконання запитів до Native API. У нашій системі доступ до всіх Плагінів відбувається централізовано через клас-одинак Framework.

Отже, у даному підрозділі дисертації описано модель взаємодії із "рідними" API у розроблюваній системі. За основу взято модель, що використовується для тих самих цілей, у React Native та адаптовано її до

нашого середовища виконання. Не зважаючи на те, що системи досить схожі вони мають відмінності. По-перше, виконання тексту програми відбувається в одному із середовищ цільових платформ. Тому до «Плагінів», що відповідають за взаємодію із API браузера, ми доступаємося оминаючи канал комунікації. По-друге, доступ до «Плагінів» відбувається централізовано через клас-одинак.

2.6. Висновки до розділу 2

У даному розділі було детально розглянуто підхід до використання WebView в якості середовища для виконання користувацьких додатків на мобільних пристроях. Також, систему було спроектовано таким чином, щоб додаток також можна було виконувати у веб-браузері на настільних комп’ютерах. На відміну від таких платформ, як React Native, PhoneGap та Flutter, які не мають такої можливості.

У загальному вигляді підхід можна представити, як показано на рис. 9. Система містить дві сторони: веб та цільова платформа. Особливістю є те, що користувацький додаток виконується на стороні веб. На стороні цільової платформи необхідно розмістити компонент WebView, написати класи для

взаємодії із API цільової платформи, відкрити канал комунікації та надіслати запит на реєстрацію "рідних" API через нього. На стороні веб необхідно також відкрити канал для комунікації, зареєструвати "рідні" модулі та надати інтерфейс для взаємодії із ними. Сторона веб може мати окремі класи / інтерфейси для взаємодії із кожним "рідних" модулем окремо або ж надавати централізований доступ до реєстру модулів напряму.

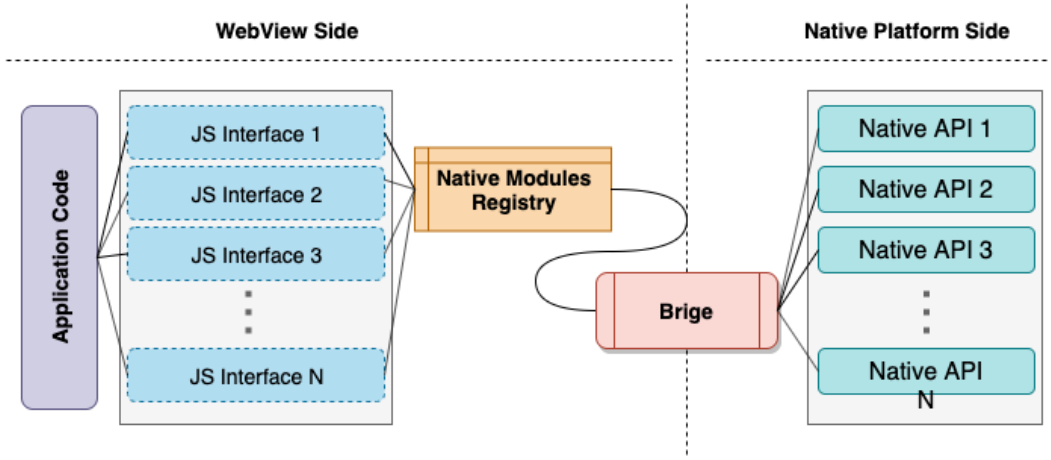


Рис. 9. Загальна архітектура підходу

Модифікація підходу полягає у тому, що у розроблюваній системі було використано допоміжний рівень у вигляді React Native. Тому розроблювана система складається з двох однотипних ланок, показаних на рис. 9, як об’єднання рис. 7 та рис. 8. Не дивлячись на те, що створюється додатковий процес, який навантажує систему, така система є доступною для розробників у сфері веб та наразі охоплює найбільшу частину IT розробки програмного забезпечення. Оскільки на сьогоднішній день бібліотеки, написані під React Native, покривають найбільшу кількість "рідного" API порівняно з іншими аналогічними системами. Також, створення нового "рідного" модуля є стандартизованим і є доступним для веб-розробника, який не є спеціалістом у Objective-C та Java. Слід не забувати про величезну спільноту та стан активного розвитку цієї технології, що свідчить про її актуальність.

Також модифікацією можна вважати сприйняття веб-середовища, як ще однієї цільової платформи. Адже, взаємодія із платформо-орієнтованою частиною тексту програми відбувається у тому ж процесі, а не через допоміжний канал комунікації. А така взаємодія має відмінності від стандартного підходу.

Слід зазначити, що для втілення модифікацій та створення каналу комунікації було проведено інтеграцію із бібліотекою mole-rpc. Про інтеграцію та проектні рішення детальніше описано у розділі 3 даної дисертації.

3. ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ РОЗРОБЛЕНИХ ПРОГРАМНИХ МОДУЛІВ МОДИФІКОВАНОГО СПОСОБУ ВИКОРИССТАННЯ WEBVIEW

Даний розділ дисертації присвячений детальному опису розроблених програмних модулів. Розроблювана система має такі основні складові, як:

– веб-додаток;

– мобільні додатки;

– компонента WebView всередині React Native;

– канал комунікації між веб та React Native сторонами

– плагіни;

– менеджер плагінів.

3.1. Платформо-орієнтовані додатки

Розроблювана система має за мету створення додатків під три платформи з єдиною кодовою базою. Цими платформами є веб, iOS та Android. Під кожну з платформ генерується окремий додаток.

Більша частина розробки припадає на веб-додаток. Адже він вміщує в себе користувацький інтерфейс, бізнес-логіку та комунікацію із віддаленим сервером.

3.1.1. *Веб-додаток та Webpack*

Веб-додаток для створюваної системи розроблюється так само, як і звичайний веб-додаток для браузера. Але з урахуванням, що доступ до всіх API браузера має відбуватися через «Плагіни». Тому що API браузера є частиною середовища виконання додатка і не буде доступним на інших платформах.

Існує два способи запуску JavaScript у браузері. По-перше, завантаження окремих .js файлів для різних частин функціональності. Це рішення важко масштабувати, оскільки завантаження занадто великої

кількості .js файлів може спричинити вузьке місце. Другий варіант – використовувати один великий .js файл, що містить увесь текст програми вашого проекту. Але це призводить до проблем у обсязі, розмірі, читабельності та підтримці [19].

Цю проблему вирішують статичні модулі, що перетворюють дерево взаємозалежних класів в один файл. Для розроблюваної системи було використано webpack.

Webpack – це статичний постачальник модулів для сучасних програм JavaScript. Коли webpack обробляє програму, він внутрішньо будує графік залежностей, який відображає кожен модуль, необхідний проекту, і генерує один або кілька вихідних пакетів – .js файлів.

Webpack – це інструмент, який дозволяє комплектувати програми JavaScript. Він може бути розширений для підтримки багатьох різних активів, таких як: зображення, шрифти та таблиці стилів [19].

У нашій розробці було застосовано додаткові плагіни, що наведено в табл. 3.

Таблиця 3

Застосовані плагіни Webpack

Назва	Призначення
copy-webpack-plugin	Копіювання окремих файлів або цілих каталогів в каталог збірки.
moment-locales-webpack-plugin	Вилучення невикористані локалізації Moment.js під час створення збірки.
webpack-bundle-analyzer	Візуалізація розміру вихідних файлів збірки за допомогою інтерактивної масштабованої карти.
NamedModulesPlugin	Відображення відносного шляху до модуля під час розробки.
DefinePlugin	Дозволяє створити глобальну константу, яку можна налаштувати під час компіляції.

Продовження табл. 3

HotModule ReplacementPlugin	Заміщення, додавання або видалення модулів під час роботи програми, що не потребує перезавантаження всієї сторінки. Використовується для під час розробки.
clean-webpack-plugin	Очищення каталогу збірки перед наступною збіркою.
optimize-css-assets-webpack-plugin	Пошук активів CSS під час компіляції webpack та їх оптимізація / мінімізація.
terser-webpack-plugin	Мінімізація тексту програми на мові JavaScript для збірки.

Також, досить часто використовують плагіни CommonsChunkPlugin або SplitChunksPlugin для оптимізації швидкості завантаження веб-сторінки. Але в нашому випадку їх використання є недоцільним. Причини описані нижче у пункті 3.1.2.

3.1.2. *Мобільні додатки та WebView*

Щодо мобільних додатків для iOS та Android, у кінцевому результаті ми отримуємо "рідні" додатки, які розміщуються на серверах дистриб'юторів додатків для смартфонів: AppStore та PlayMarket.

Мобільні додатки запускаються у своєму рідному середовищі виконання (Objective-C / Java) та інтерпретують текст програми, що знаходиться на стороні React Native і написаний на JavaScript. На стороні ReactNative посилається запит до "рідної" частини додатка, щоб відкрити веб-переглядач – WebView. Із цим запитом до веб-переглядача завантажується текст програми на мові HTML сторінки для створення дерева елементів і текст програми на мові JavaScript скомпільованого веб-додатка для відображення інтерфейсу користувацького додатка, виконання бізнес-логіки.

Вимогою API компонента WebView є те, щоб завантажуванні HTML- та текст програми на мові JavaScript були у вигляді рядка. Це і є причиною чому для компіляції веб-додатка не було використано плагіни для

розділення збірки на частини. У лістингу програми 5 показано, як використовується компонент WebView.

Лістинг 5. Використання компонента WebView

```
import React, { Component } from 'react';
import PropTypes            from 'prop-types';
import { WebView }          from 'react-native-webview';

import HTML    from 'external-html.js';
import myJSLib from 'external-lib.js';

import styles from './WebViewStyles.js';

export default class WebViewWrapper extends Component {
    render() {
        const webViewProps = {
            source : { html: HTML }
            injectedJavaScript : myJSLib
        };

        return (
            <WebView
                {...webViewProps}
                originWhitelist    = {[ '*' ]}
                style              = {styles.layout}
                bounces            = {false}
                domStorageEnabled
                javaScriptEnabledAndroid
                javaScriptEnabled
            />
        );
    }
}
```

3.2. Канал комунікації

Канал комунікації реалізовано на основі клієнт-серверної архітектури. Клієнт та Сервер у даній системі реалізовано з використанням бібліотеки *mole-rpc*. Вона надає інтерфейс, що не залежить від транспорту та його реалізації. Також, сервер може мати декілька транспортів одночасно. Повідомлення передаються у форматі JSON RPC.

Розроблений для даної системи транспорт базується на вбудованому в браузер генераторі подій – *onmessage*. Для цього було використано такі інтерфейси, як: глобальна змінна *window* та її метод *onmessage* у середовищі WebView і компоненту від React Native <*WebView*>, посилання на нього та його властивість *onMessage*.

Також, слід зазначити, що для обробки потоку подій було застосовано шаблон проектування Pub/Sub. Де в ролі видавця виступають глобальна змінна *window* та компонента *<WebView>*, а підписниками екземпляри серверів на кожній із сторін.

Повідомлення передаються у форматі JSON RPC. Опис полів для Запиту та Відповіді наведено у табл. 4.

Таблиця 4

Опис полів запитів JSON-RPC

Поле	Значення
Запит	
jsonrpc	версія JSON RPC
id	унікальний ідентифікатор запиту
method	Один із методів, що зареєстровані на сервері методом *expose*
params	параметри для виконання операції на сервері
Поле запиту params[1]	
pluginId	унікальній ідентифікатор плагіну запит, на який виконується
method	один із доступних методів плагіну
args	параметри для виконання запиту на плагін
Відповідь	
jsonrpc	версія JSON RPC
id	унікальний ідентифікатор запиту
result	дані результату виконання запиту

3.2.1. *Middleware*

За допомогою middleware, який є основною складовою каналу комунікації, Клієнт та Сервер можуть обмінюватися повідомленнями. Для

[1] Структура для цього поля стандартизована лише в контексті даної роботи. Іншими розробниками може використовуватися будь-яка зручна для них структура цього поля.

комунікації між сторонами веб та React Native було написано власний middleware для інтеграції з бібліотекою mole-rpc. А для надання можливості й іншим розробникам користуватися цим middleware, його було оформлено у бібліотеку та викладено до NPM – реєстру програмного забезпечення.

На рис. 10 зображено клас middleware. Як вимагає API бібліотеки mole-rpc, middleware реалізує два методи: onData та sendData. Перший використовується для сповіщення об'єкта Сервера або Клієнта про отримання повідомлення. А другий – для генерування повідомлень Клієнтом.

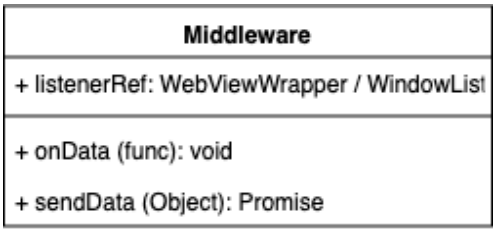


Рис. 10. Клас Middleware

Також, клас middleware має поле listenerRef, що зберігає у собі посилання на один із вбудованих генераторів подій. І активно використовується у методах onData і sendData для пересилання відповідних типів повідомлень.

3.2.2. *WebViewWrapper та WindowListener*

Для того, щоб транспорт для mole-rpc змогли використовувати й інші розробники, до описаної вище бібліотеки було додано компонент WebViewWrapper та клас WindowListener.

На рис. 11 схематично зображено поля та методи, якими доповнюють та які надають створені структури вбудовані.

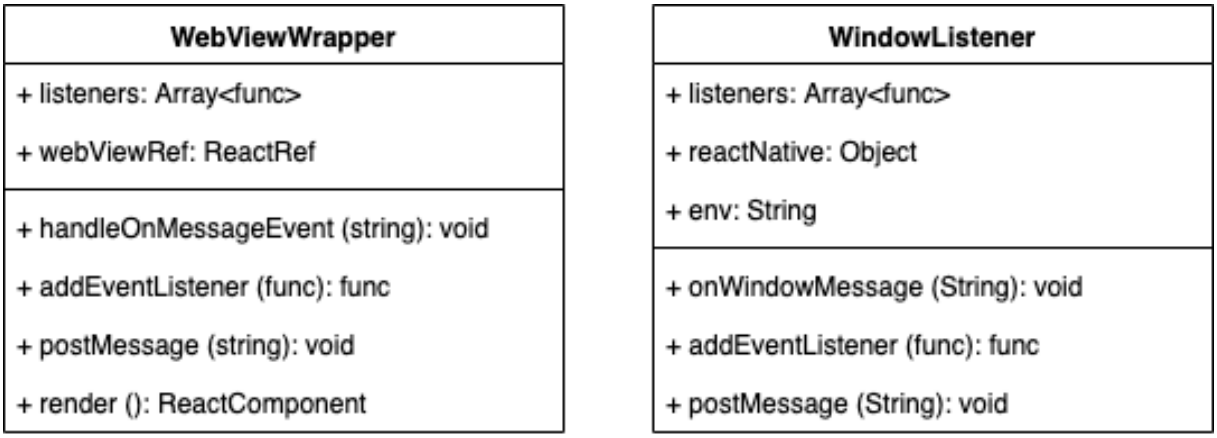


Рис. 11. Компонент WebViewWrapper та клас WindowListener

Створені структури є схожими, але призначені для різних платформ і середовищ виконання, тому мають свої особливості.

Обидва класи мають поле *listeners* – слухачі. Воно містить масив функцій зворотного виклику, які необхідно викликати, коли надходить повідомлення. Виклики слухачів відбуваються всередині *handleOnMessageEvent* та *onWindowMessage* методів класів. Ці методи також являються функціями зворотного виклику, але реєструються у вбудованих у платформу генераторах подій. Параметром, який передається при виклику функції слухача є рядок повідомлення.

Додати слухача можна викликавши метод *addEventListener* та передавши функцію зворотного виклику першим аргументом. Метод *addEventListener* повертає функцію, яка своїм виконанням видаляє функцію зворотного виклику із масиву слухачів.

Описані вище структурні компоненти відповідають за функціональність отримання повідомлень, а для їх надсилання використовуються поле *webViewRef* / *reactNative* та метод *postMessage*. Поля *webViewRef* та *reactNative* містять посилання на об’єкти глобальних генераторів подій для кожної з платформ відповідно. А метод *postMessage* екранує такий самий метод глобальних генераторів подій назовні.

3.2.3. *Клас каналу комунікації*

Для зручності користування Сервером і Клієнтом від mole-rpc та створеним транспортом було створено клас, що поєднує ці три структурні компоненти системи – канал комунікації. На рис. 12 зображено поля та методи класу каналу комунікації для кожної із сторін.

Як було описано у підрозділі 2.4 цієї дисертації кожна із сторін має бути як Сервером, так і Клієнтом. Тому класи каналів комунікації містять посилання на екземпляри відповідних структур. Також, кожен з класів містить метод *init*, в якому відбувається реєстрація у розробленому транспорті та глобальних генераторах подій слухача від класу MoleServer.

У кожному з класів каналів комунікації також присутній метод execute. За допомогою цього метода відбувається генерування повідомлення для відправки на іншу сторону. А саме, всередині цього метода відбувається виклик методу *callMethod* класу MoleClient. Першим параметром передається рядок «execute» – ідентифікатор зареєстрованого у MoleServer методу. Другим параметром передається об’єкт, який буде поміщено у поле запиту – *params*.

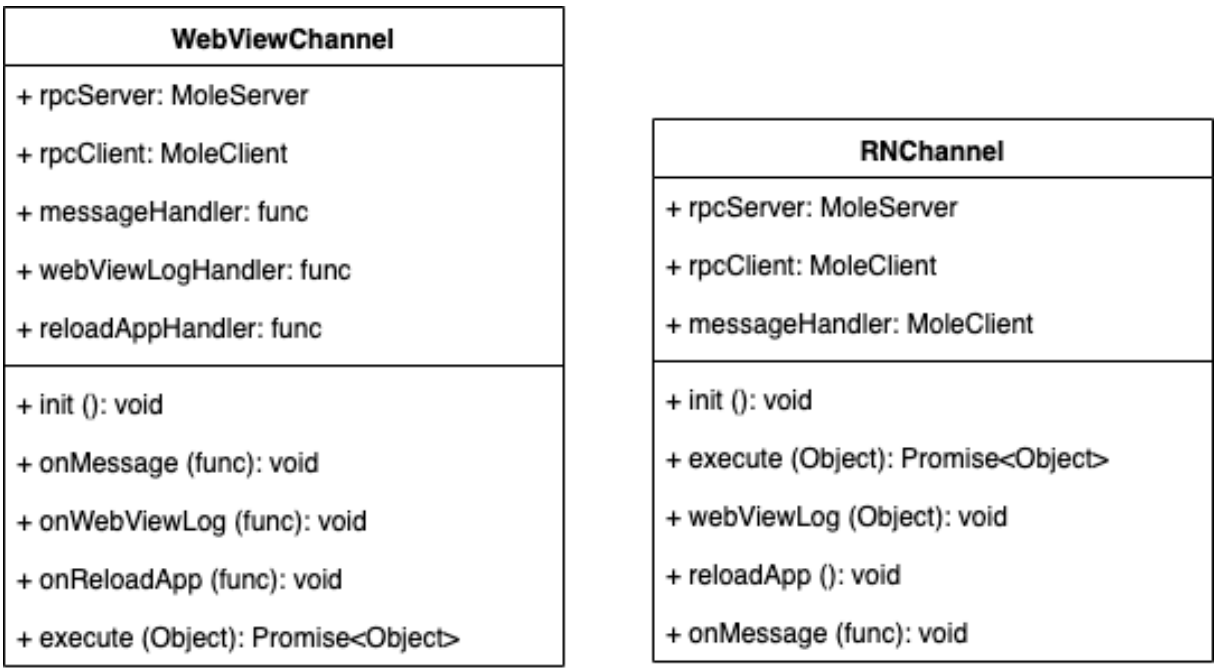


Рис. 12. Класи каналу комунікації на сторонах React Native та веб

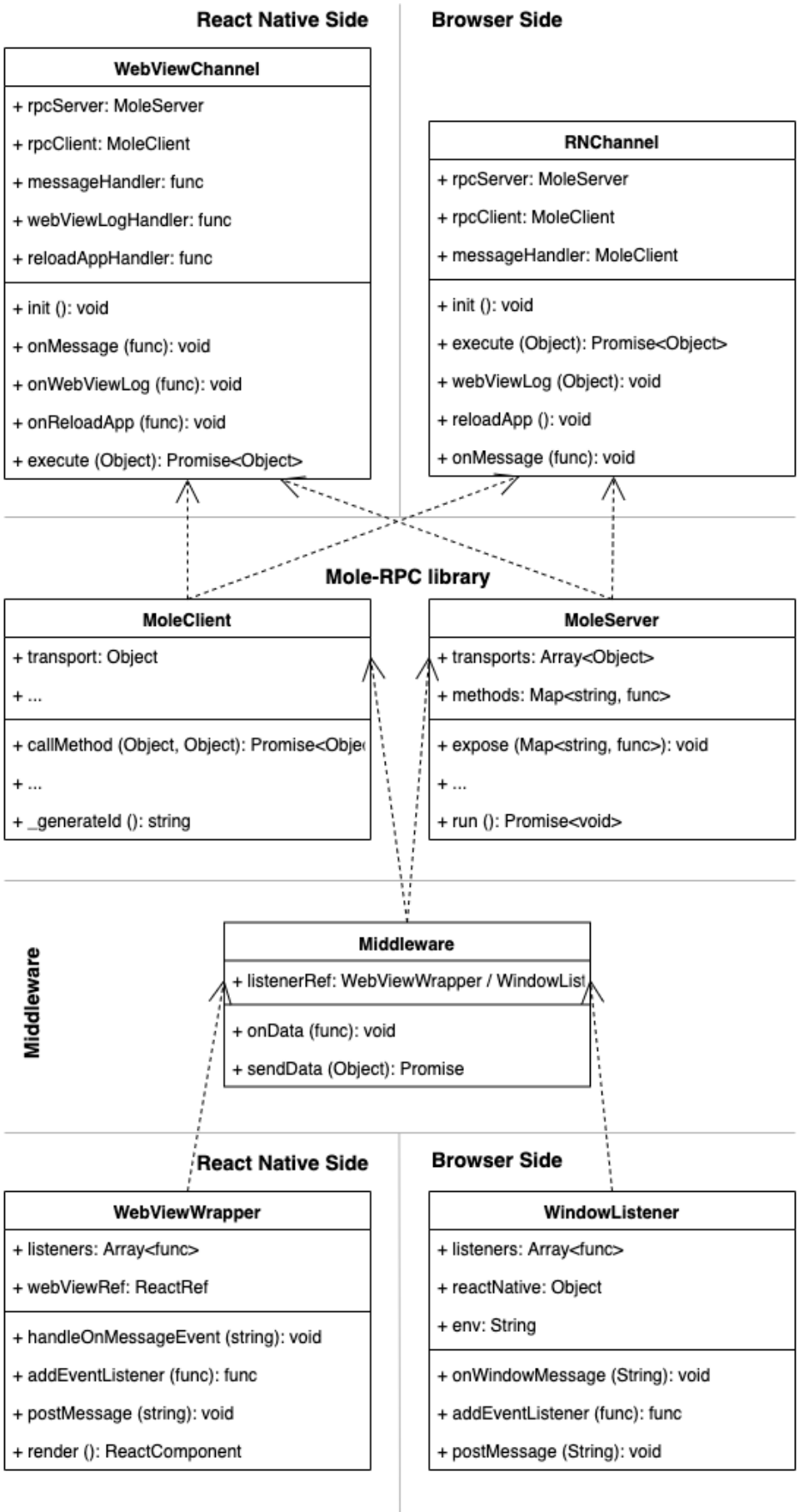


Рис. 13. Діаграма класів каналу комунікації

Ще одним однаковим методом є *onMessage*. Цей метод виконує роль сетера обробника подій типу «execute», що були зареєстровані на Сервері. Передана першим параметром функція зворотного виклику буде записана у поле *messageHandler* та викликана із надходженням відповідного повідомлення на Сервер. Те саме призначення виконують методи *onWebViewLog* та *onReloadApp* і поля *webViewLogHandler* та *reloadAppHandler*. Вони відповідають за обробку подій від Сервера «webViewLog» й «reloadApp» відповідно.

На стороні ж клієнта, тобто у каналі на стороні браузера, реалізовано методи для генерації подій типів «webViewLog» й «reloadApp» – *webViewLog* і *reloadApp* відповідно. Вони працюють за аналогією до методу *execute* цих класів.

Якщо відобразити всі складові каналу комунікації у розроблюваній системі разом, отримаємо діаграму класів зображену на рис. 13.

3.3. Плагіни

Плагіни у даній системі виконують роль архітектурних компонент, зорієнтованих на специфічне для платформи середовище виконання. Це зумовлено тим, що WebView не має повного API звичайного браузера [13]. Наприклад, WebView не має доступу до API *localStore* або *browserHistory,* що є невід'ємною частиною будь-якого веб-додатку. З іншого боку, мобільні додатки мають більші можливості щодо роботи з API операційної системи смартфону, аніж браузери. Наприклад, отримання доступу до списку контактів, камери, галереї, детальної інформації про інтернет з'єднання, файлової системи та інше [13].

Таким чином, для кожного середовища виконання додатку необхідно написати специфічний плагін. За допомогою менеджера плагінів до нього можна звернутися з будь-якого місця додатку і не замислюватися над тим, в якому середовищі зараз виконується текст програми.

Плагіни можуть бути двох типів: утиліти та компоненти. Утиліти – це класи для роботи з API, наприклад, локальним сховищем або інформацією про стан інтернет з'єднання смартфону. Компоненти – це ті плагіни, які мають графічне представлення. Наприклад, камера, модальне вікно вибору дати та часу, відео-плеєр тощо.

3.3.1. *Обов'язкові властивості*

Обидва типи плагінів мають містити статичне поле *pluginOptions*, що відображає тип плагіна та його унікальний ідентифікатор. Також кожний з плагінів отримує функцію зворотного виклику *onInitEnd*. У випадку плагіна-утиліти ця функція буде передана до класу у конструкторі, для плагіна-компоненти – через поле властивостей. Функція *onInitEnd* має бути викликана після ініціації плагіна. Її призначення зареєструвати у менеджері плагінів публічні методи плагінів для виклику ззовні.

У лістингу 6 наведено шаблони для створення плагінів обох типів.

Лістинг 6. Шаблони для створення плагінів

```
function
ComponentPlugin(props) {
    const { onInitEnd }
= props;

    const _method1 = ()
=> {
        // some stuff
    };

    render() {
        return (
            // component
visualization
        );
    }
}

ComponentPlugin.propType
s = {
    onInitEnd :
PropTypes.func.isRequire
d
};

ComponentPlugin.pluginOp
tions = {
```

```
class UtilPlugin {
    constructor({ onInitEnd }) {

onInitEnd(UtilPlugin.pluginOptio
ns.id, {
            publicMethodName :
this._method1
        });
    }

    static pluginOptions = {
        id   : 'plugin-util-id',
        type : 'util'
    };

    _method1 = () => {
        // some stuff
    }
}

export default UtilPlugin;
```

Продовження лістингу 6

```
    id : 'componeent-
plugin-id',
    type : 'component'
};

export default
ComponentPlugin;
```

3.3.2. *Внутрішній стан плагіна*

Також слід виділити спеціально створені допоміжні функції, що надають плагінам функціональність маніпулювання станом. Вони були розроблені для стандартизації сповіщень користувацького додатка, який знаходиться всередині WebView про зміни внутрішнього стану плагіна або платформного API з яким цей плагін комунікує. Наприклад, сповіщення про зміну типу інтернет з'єднання з WiFi на мобільний інтернет, яке ініційоване операційною системою смартфона.

За основу було взято схожу функціональність компонентів класу Component від React.

Робота із станом реалізована за рахунок замикання. Тому маніпулювання станом можливе лише через спеціальні функції, що повертаються в результаті виклику функції-помічника.

Основною задачею розроблених функцій-помічників є маніпулювання внутрішнім станом та надсилання сповіщень про зміни на сторону веб-додатка. Таке сповіщення має чітко визначені поля для структури, що описана у табл. 4 цього розділу. Які саме, описано в табл. 5.

Таблиця 5

Поля сповіщення про зміни внутрішнього стану плагіна

Поле	Значення
pluginId	system
Method	updateState
Args	{ pluginId, prevState, newState }

Розроблені функції-помічники мають таку сигнатуру. На вхід у параметрах вони приймають змінну типу Object, яка має поля: *id* та *state*, де перше поле – це унікальний ідентифікатор плагіна, а друге – початковий внутрішній стан. Повертають такі функції-помічники екземпляр типу Object, який містить посилання на такі функції: *getState*, *updateState*, *resetState*.

Функція *getState* не приймає аргументів і повертає поточне значення внутрішнього стану плагіна.

Функція *updateState* приймає першим аргументом змінну типу Object, яка містить мапу полів стану, що треба змінити. Ключ – це назва поля, а значення – значення, що необхідно встановити у відповідне поле внутрішнього стану. Під час виконання саме цієї процедури виконується відправлення сповіщення на сторону веб про оновлення внутрішнього стану плагіна.

Функція *resetState* встановлює внутрішнє значення стану у початкове значення та сповіщає систему про відповідні зміни у плагіні.

Реалізацію таких функцій-помічників для відповідних типів плагінів наведено у Додатку 1.

3.3.3. *Реєстрація плагінів*

Реєстрація плагінів у менеджері плагінів відбувається під час запуску додатка. Реєстрація плагінів-утиліт відбувається під час ініціювання Менеджера Плагінів. Адже вимагає лише створення екземпляра класу. Стосовно плагінів-компонентів маємо зауважити, що процес реєстрації є складнішим. Це зумовлено тим, що такі плагіни мають графічне представлення, тому мають бути додані до дерева інтерфейсу DOM (Document Object Model).

Цю проблему було вирішено за рахунок використання такої можливості React, як HOC (High Order Component) – компонент високого порядку.

Для розроблюваної системи було створено HOC – injectPlugins. Його призначення відобразити вміст сторінки, що передано у параметрах, та на тому ж рівні відобразити плагіни-компоненти із абсолютним позиціюванням. Цей допоміжний компонент має доступ до класу Менеджера Плагінів й отримує від нього список компонентів для відображення. Текст програми компонента високого рівня injectPlugins наведено у лістингу 7.

Лістинг 7. HOC injectPlugins .js

```
import React, { Component } from 'react';
import { View }             from 'react-native';

import pluginManager from '../
pluginManager';



function injectPlugins(WrappedComponent) {
    return class extends Component {
        render() {
            const componentPlugins =
pluginManager.getComponentPluginsWithProps();

            return (
                <View style={{ flex: 1 }}>
                    <WrappedComponent
                        {...this.props}
                    />
                    {componentPlugins.map(({ default: Plugin,
                        ...otherProps }) => (
                        <Plugin
                            {...otherProps}
                            key        = {Plugin.pluginOptions.id}
/>
                    ))}
                </View>
            );
        }
    };
}

export default injectPlugins;
```

3.4. Менеджер плагінів

Менеджер плагінів в нашій системі – це клас, який володіє реєстром всіх доступних плагінів, та функціональністю їх додавання, видалення та доступу до них. Екземпляр такого класу створюється як на стороні React Native, так і на стороні веб розроблюваної системи. Також, слід зазначити,

що всі плагіни, які наявні для "рідних" платформ, має сенс продублювати і для веб. Це робиться в рамках сумісності тексту програми для виконання у всіх трьох середовищах. У випадку, коли створюється, наприклад, плагін, що відповідає за визначення геолокації користувача додатка, ми маємо різні API для трьох платформ. У випадку, коли ми реалізуємо плагін, що відповідає за відслідковування стану інтернет з'єднання, ми не маємо відповідного API в браузері, але маємо звернення до відповідних плагінів у користувацькому додатку. Тому є сенс створити плагін із даними за замовчуванням, які не змінюються.

Клас менеджера плагінів присутній на обох сторонах розроблюваної системи, але його реалізації мають важливу відмінність. Менеджер плагінів на стороні веб надає перевагу плагінам, що надходять зі сторони платформи. Це дозволяє створювати плагіни на стороні веб з тими самими унікальними ідентифікаторами, як і на платформах. І за умови виконання тексту програми користувацького додатку на платформі смартфону менеджер плагінів на стороні веб перевизначить відношення ідентифікатора з екземпляра веб-плагіна на платформо-орієнтований екземпляр.

На рис. 14 зображено поля та методи класу менеджера плагінів.

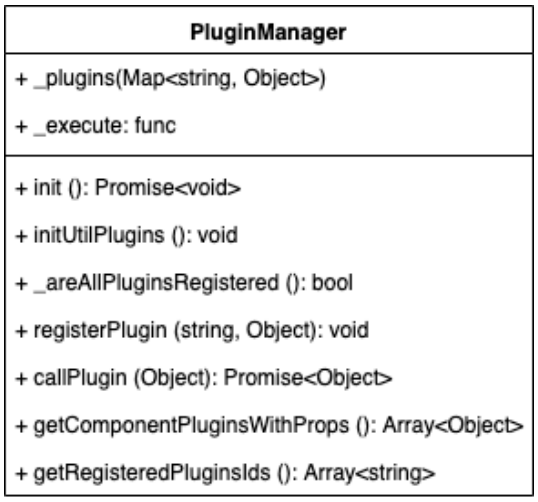


Рис. 14. Клас PluginManager

Він містить поле *_plugins*. Це мапа ключами якої є унікальні ідентифікатори доступних плагінів, а значенням об'єкт, що складається з двох полів: *status* і *methods*. Де *status* може приймати одне із значень: PENDING, INITIALIZED. А *methods* є мапою з ключами – назвами доступних методів у плагінів, значеннями – посиланнями на функції для виклику.

Поле *_execute* містить посилання на метод *execute* класу комунікації описаного у пункті 3.2.3.

Метод *init* всередині викликає метод *initUtilPlugins*, який створює екземпляри класів для плагінів-утиліт, та чекає, поки всі заявлені плагіни не буде ініційовано. Сигналом, що плагін ініційовано слугує виклик зворотної функції, що передається до екземпляра плагіна разом із властивостями під назвою *onInitEnd*. Цією функцією зворотного виклику є метод менеджера плагінів – *registerPlugin*.

Оскільки клас менеджера плагінів по своїй суті є реєстром плагінів, взаємодія із плагінами виконується через метод його інтерфейсу – *callMethod*.

Цей метод перевіряє наявність плагіна із яким є намір провести взаємодію у реєстрі та наявність зареєстрованого метода для цього плагіна. У разі відсутності необхідних даних, метод *callMethod* генерує відповідну помилку. Інакше, чекає результату виконання звернення до відповідного плагіна і повертає результат.

3.5. Висновки до розділу 3

Отже, модифікований підхід використання WebView для розроблення мобільних додатків було розроблено за рахунок створення та використання таких структурних компонент, як:

– Компонент відображення користувацького інтерфейсу – WebView, який доступний переважно на всіх сучасних платформах, включно на iOS й Android.

– Канал комунікації між React Native та WebView, що базується на основі клієнт-серверної архітектури та вбудованих механізмах глобальних генераторів подій у браузері. Канал комунікації має декілька складових таких як, Клієнт, Сервер, Middleware, класи-обгортки для глобальних генераторів подій на обох сторонах та клас, які поєднали в собі всі складові.
– Плагіни, що забезпечують доступ до платформо-залежних API. Вони надають однаковий інтерфейс, незалежно від того, на якій з платформ зараз запущено додаток. Тим самим звільняючи розробника від перевірок на кожному кроці типу поточного середовища виконання.
– Менеджер плагінів, який є відповідальним за роботу з різними типами плагінів (утилітами й компонентами), їх зберігання у реєстрі та забезпечення доступу до них.

Завдяки їх взаємодії стало можливим, щоб веб-додаток працював із платформними API, які надають операційні системи смартфонів так, як з "рідними" мобільними додатками. Також не буде втрачено сумісність із веб-середовищем.

4. АНАЛІЗ РОЗРОБЛЕНИХ ПРОГРАМНИХ МОДУЛІВ

4.1. Тестування розробленого програмного забезпечення

Розроблювану систему можна розділити на такі частини для тестування, як:

– Транспорт між сторонами WebView та React Native.

– Плагіни для кожної з платформ (веб, iOS, Android).

– Менеджер плагінів на кожній із сторін (веб, React Native).

У процесі розробки було виконано тестування за такою методикою:

1. Модульне тестування.

2. Інтеграційне тестування.

3. Системне тестування.

4. Регресивне тестування.

На етапі модульного тестування було створено автоматизовані unit-тести для плагінів-утиліт кожної з платформ: локального сховища даних (LocalStoragePlugin), інформації про стан інтернет-з'єднання (NetInfoPlugin), інформації про дані девайса (DeviceInfoPlugin), геолокації (GeolocactionPlugin). Для плагінів-компонентів було проведено ручне тестування.

Також, на етапі модульного тестування було проведено тестування класу транспорту за визначеними у бібліотеці mole-rpc автоматичними тестами [17].

На етапі інтеграційного тестування було відлагоджено взаємодію класу Транспорту із класами WindowListener / WebViewWrapper.

Також, на етапі інтеграційного тестування було відлагоджено взаємодію сторін React Native та веб. Тестування було проведено за підходом «знизу вверх».

На етапі системного тестування було проведено функціональне тестування методом «білого ящика» та низка декілька тестів: тестування безпеки, тестування на сумісність.

Під час функціонального тестування для окремих полів було проведено:

– позитивне тестування (коректні кроки, коректні дані);

– негативне тестування (введення некоректних даних).

Тестування на сумісність було проведено з використанням тестового оточення наведеного в табл. 6.

Таблиця 6

Тестове оточення

Мобільні телефони	iPhone 5s, Meizu M5 Note, Xiaomi A1, Prestigio, Samsung, LG, Huawei P Smart+
Операційні системи	iOS, Android
Роздільна здатність екрану	320x568; 375x667; 4141x736; 360x640; 1080x1920
Браузери	Edge, Google Chrome, Safari, Firefox

Тестування проводилося ітераційно. Після кожного тесту знайдені помилки було виправлено.

Слід зазначити, що під час розроблення системи проводилося регресивне тестування, що включає в себе автоматизовану перевірку тексту програми з використанням таких утиліт: як: ESlint для тестування JavaScript-файлів й StyleLint – файлів стилів (*.css, *.less). І автоматизовану перевірку безпечності встановлених сторонніх бібліотек з використанням npm audit.

За результатами останнього тесту 94% запланованих юзкейсів відпрацювали добре. Тому дана версія продукту придатна для залучення бета-тестувальників.

4.2. Аналіз розробленого програмного забезпечення

Після завершення роботи над модифікованим підходом використання WebView для розроблення мобільних додатків, його було проаналізовано на відповідність користувацьким вимогам описаним у висновках до першого

розділу цієї дисертації у підрозділі 1.5, аналіз якості тексту програми та проведено заміри часу за який проходить запуск додатка залежно від кількості підключених плагінів.

Розроблений підхід:

– використовує WebView для виконання у ньому тексту програми, що відповідає за відображення користувацького інтерфейсу та бізнес-логіку;
– має стандартизовану архітектуру для взаємодії із "рідними" API платформи;
– має базову обгортку для мобільних платформ у вигляді додатка на React Native, що виступає транслятором: для тексту програми клієнтського додатку надає єдиний API для взаємодії з системними сервісами і виконує необхідні команди API залежно від платформи, на якій зараз виконується програма;
– виконується як на мобільних пристроях Android, iOS, так і у веб-переглядачі.

Слід також зазначити, що додаток написаний на React Native може бути скомпільований для таких платформ, як: Universal Windows Platform, Windows Presentation Foundation для підтримки Windows Phone та настільних комп'ютерів з ОС Windows, DOM для підтримки вебу, Desktop для підтримки настільних комп'ютерів з ОС Ubuntu, macOS – настільні комп'ютери з ОС MacOS, tvOS – адаптація під Apple tvOS [21]. Тому розроблений метод із базовою технологією React Native має перспективи, щодо збільшення покриття сучасних девайсів з використанням однієї кодової бази.

Однією з важливих метрик платформи є швидкість ініціалізації додатка. У розробленій системі на цю метрику може впливати кількість підключених "різних" плагінів. Результати замірів часу наведено на графіку на рис. 15.

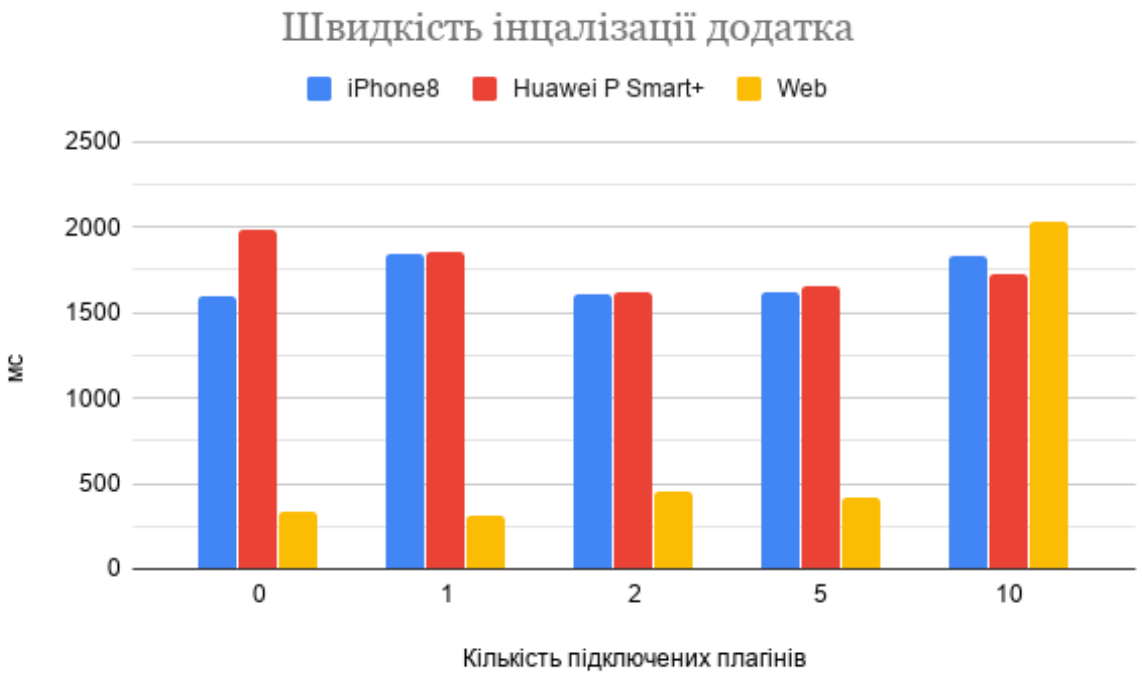


Рис. 15. Графік залежності швидкості ініціалізації додатка залежно від кількості плагінів та цільової платформи

Як видно з графіка швидкість ініціалізації додатка не залежить від кількості підключених плагінів для мобільних платформ. Чого не можна сказати про веб-додаток. Але такі результати можна вважати задовільними, адже при роботі з веб-додатком не має необхідності використовувати одразу всі плагіни. Також, текст програми, що завантажується для веб-додатка може бути подрібнено на частини та завантажено поступово. Таким чином буде пришвидшено "час першої взаємодії" користувача із додатком.

Ця метрика є важливою та входить у перелік тих, які використовують для аналізу продуктивності сайтів зокрема. Максимальне допустиме значення цієї метрики сягає 5-15 секунд [22]. На нашому графіку видно що для жодної з платформ ця метрика не перевищує норму, що є задовільним результатом.

Ще однією важливою метрикою є розміри додатка. Адже, по-перше, кількість пам’яті на смартфонах обмежена. По-друге, для веб-додатка ця

метрика впливає на швидкість завантаження. На рис. 16 та рис. 17 зображено результати аналізу розміру збірок для платформ: Android та веб.

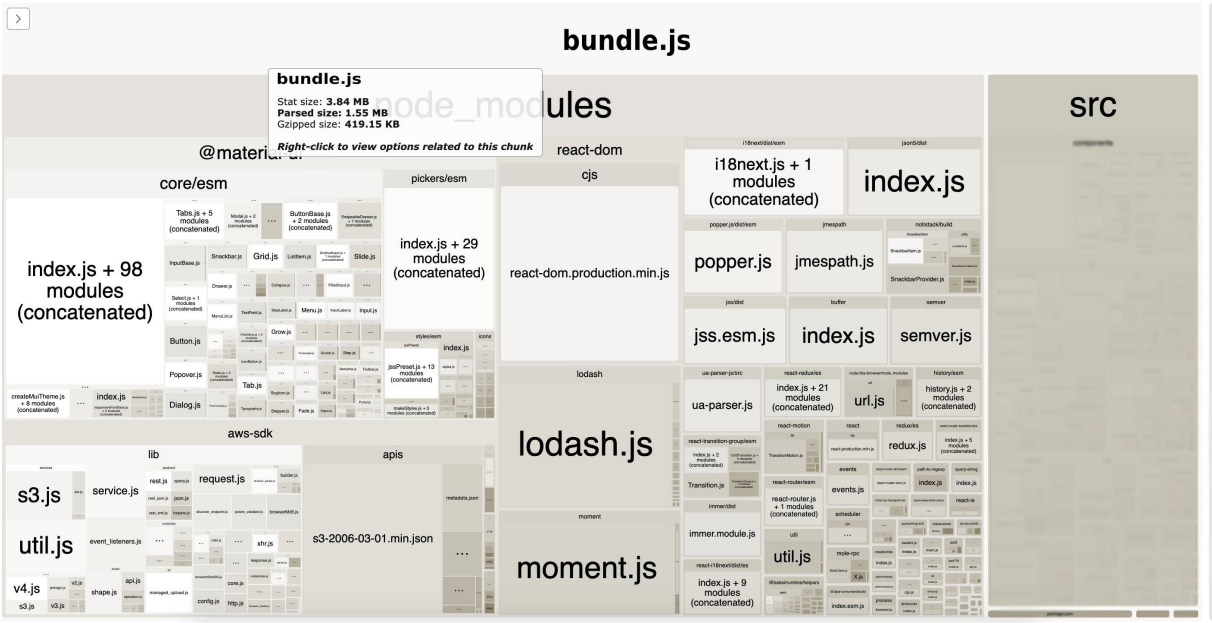


Рис. 16. Результати аналізу збірки для веб

Рекомендований розмір збірки для веб-додатка становить 244 КБ. Розроблений додаток важить 419.5 КБ, що приблизно у два рази більше. З графіка видно, що більшу частину займають сторонні бібліотеки. Тому має бути проведена робота над зменшенням їх розмірів через включення до збірки лише тієї частини тексту програми, яка необхідна, а не всієї сторонньої бібліотеки. Текст програми платформи, безпосередньо, важить 67,91 КБ, що є задовільним результатом. Але слід зауважити, що зображення, яке включено до збірки важить 21.26 КБ, що становить третину від всього тексту програми платформи. Тому воно має бути виключено із збірки та завантажуватися окремо.

Щодо платформи Android, з рис. 17 можемо побачити, що розмір завантажувального файлу становить 27.5 МБ. Слід зауважити, що додаток було зібрано у форматі *.aab.

Android App Bundle – це новий формат завантаження, який включає весь зібраний текст програми та ресурси додатка, але відкладає генерацію APK та підписання до Google Play.

Нова модель сервісу додатків Google Play під назвою "Dynamic Delivery" використовує *.aab для створення та подання оптимізованих APK-файлів для кожної конфігурації пристрою, тому вони завантажують лише текст програми та ресурси, необхідні для запуску програми [23].

Тому розмір файлу, що завантажується із Google Play має розмір у 13 МБ, замість 28 МБ.

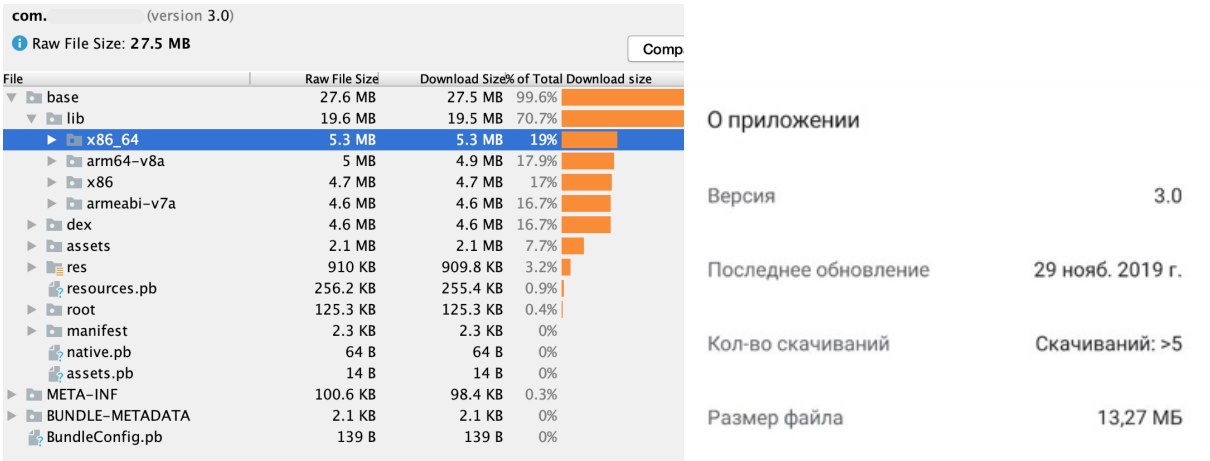


Рис. 17. Результати аналізу збірки для платформи Android

4.3. Вдосконалення розробленого програмного забезпечення

Виходячи з результатів аналізу збірок розробленої системи у попередньому розділі даної роботи, першим вдосконаленням буде зменшення розмірів збірок для кінцевих платформ за рахунок оптимізації розмірів зображень та включення до збірок лише використовуваного тексту програми сторонніх бібліотек. Це пришвидшить завантаження додатку як у веб-середовищі, так і на цільових мобільних платформах. Також, це пришвидшить роботу додатка в цілому.

Також, важливим вдосконаленням є надання можливості конфігурування списку плагінів, що використовуються. Кількість "рідних" впливає на розмір збірки та час завантаження додатка, як для веб, так і для

мобільних цільових платформ. Це питання може бути вирішено за рахунок створення модулів для webpack (веб), gradle (Android) та cocoapods (iOS).

Розроблена система складається із двох ключових складових, які можуть і мають бути опубліковані у реєстрі програмного забезпечення – NPM. Це класи транспорту каналу комунікації для бібліотеки Mole-RPC та система цілком.

Транспорт має бути оформлений у бібліотеку, яка встановлюється для кожного з проектів: веб та React Native, і надає можливість комунікації з використанням технології віддаленого виклику процедури (RPC) та повідомлень у форматі JSON.

Система у контексті даної роботи означає два додатки: веб та React Native, які вже мають налагоджений канал комунікації, функціональність для роботи із платформо-залежним текстом програми – плагінами. Вся система має бути оформлена у шаблонний додаток та завантажено на платформу GitHub в режимі публічного доступу. А базові команди для створення та конфігурування такого шаблону оформлено в консольну утиліту та опубліковано у реєстрі програмного забезпечення – NPM.

Також слід зазначити, що додаток написаний на React Native може бути скомпільований для таких платформ, як: Universal Windows Platform, Windows Presentation Foundation для підтримки Windows Phone та настільних комп'ютерів з ОС Windows, DOM для підтримки вебу, Desktop для підтримки настільних комп'ютерів з ОС Ubuntu, macOS – настільні комп'ютери з ОС MacOS, tvOS – адаптація під Apple tvOS [21]. Тому ще одним із напрямків є адаптація розробленої системи для нових платформ.

Ще одним напрямком для вдосконалення платформи є впровадження модулів віртуальної та доповненої реальності. Дані технології досить активно розвиваються зараз і досить швидко набудуть популярності. Для React Native вже створюються модулі для роботи із віртуальною та доповненою реальностями [24].

4.4. Висновки до розділу 4

У даному розділі було описано методологію тестування для розроблюваної системи, проаналізовано результати роботи та визначено напрями для подальшого вдосконалення системи у майбутньому.

За результати тестування можна заявити, що розроблена система не має помилок, які можуть призвести до виключних ситуацій при роботі користувача. Тому може вважатися готовою до залучення бета-тестувальників.

Аналіз системи дав позитивні результати щодо незалежності початкового завантаження додатка від кількості використовуваних плагінів та щодо метрики продуктивності веб-додатків – "час першої взаємодії". Але також було визначено, що система має недоліки щодо розмірів збірок, та перераховано заходи щодо усунення цієї проблеми.

Також, за результатами аналізу було виявлено, що система повністю відповідає заявленим вимогам, що були висунуті під час аналізу систем-аналогів.

В останній частині даного розділу було визначено основні напрями щодо вдосконалення розробленої системи. А саме:

– усунення недоліків, виявлених під час аналізу розробленої системи;
– надання публічного доступу до розроблених модулів через розміщення їх у якості бібліотеки, шаблону та консольної утиліти на відповідних ресурсах;
– адаптація розробленої системи до інших цільових платформ, що підтримуються технологією React Native;
– впровадження модулів віртуальної та доповненої реальності.

5. ПОБУДОВА БІЗНЕС-МОДЕЛІ

5.1. Виділення проблеми

Сьогодні для успішного ведення бізнесу, що зорієнтований на взаємодію з великою кількістю людей, необхідно або створювати власні додатки, або ж інтегруватися в існуючі системи такі, як соціальні мережі, конструктори сайтів. Створювання сучасних програмних систем вимагає розповсюдження на найбільшу користувацьку аудиторію, що складається з веб-додатку, мобільних додатків для iOS та Android.

Бізнеси зацікавлені у створенні та утриманні великої кількості постійних клієнтів. Тобто, утриманні високого рівня лояльності та наданні якісного інструменту для вирішення проблем клієнтів – користувачів електронних додатків.

Заохочення клієнтів та їх утримання можливе за рахунок надання можливості клієнтам отримувати доступ до власних даних будь-яким зручним методом. Чи за допомогою мобільного телефону, сайту, настільного додатка, чи навіть смарт-годинника.

У сфері інформаційних технологій існує конкуренція. І як наслідок існують однотипні платформи, які вимагають адаптуватися під їх специфічні стандарти. Наприклад, різні операційні системи для смартфонів або різні протоколи комунікації для девайсів інтернету речей.

Тому виникають проблеми пов’язані із створенням та підтримкою окремих додатків для багатьох платформ. По-перше, це вартість розробки, оскільки для кожної платформи необхідно винаймати команду, яка на ній спеціалізується. По-друге, великі часові затрати на комунікацію між командами розробки та налагодження відповідності між платформними додатками. Ці фактори також призводять до втрати гнучкості розробки та збільшення ціни внесення змін на будь-якому етапі.

Для повноти розуміння зазначених проблем побудуємо дерево проблем, на якому зображено конкретні причини та наслідки труднощів, з

якими користувач та бізнес можуть зустрітися при ігноруванні проблем. Дерево проблем наведено на рис. 18.

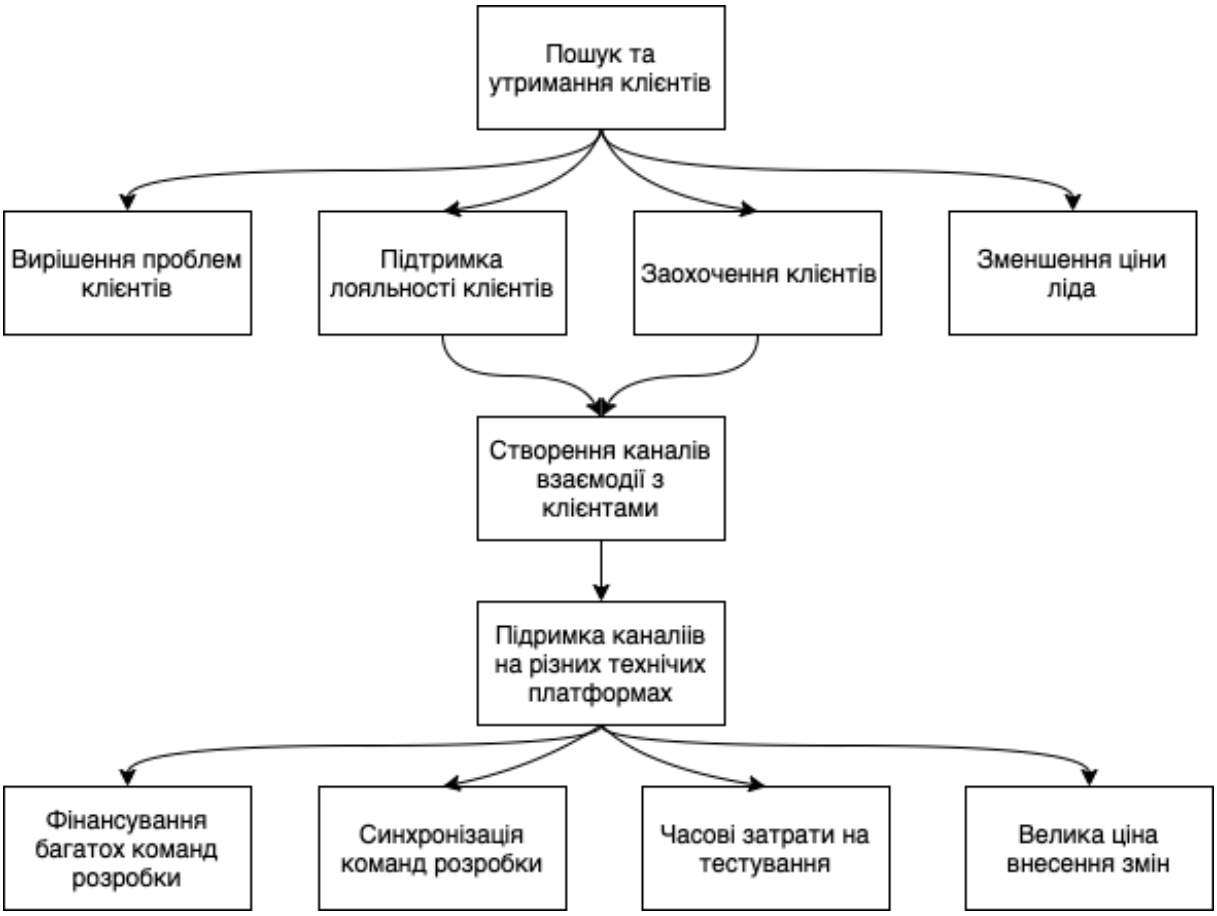


Рис. 18. Дерево проблем

Сьогодні все більше бізнесів звертаються до розробників для створення мобільних додатків, що використовують один і той самий текст програми для всіх платформ. У тому числі, такий текст програми може бути перевикористано для створення веб-додатків. Крос-платформність додатків надають технології: ReactNative, Ionic, PhoneGap, Xamarin, Flatter і т.д. Більшість з них базується на JavaScript.

5.2. Зацікавлені сторони

В процесі створення бізнес-моделі проекту було сформовано групи зацікавлених в реалізації проекту сторін:

– власники користувацьких додатків (бізнеси);

– проектні менеджери;

– розробники (спільнота та компанії з розробки);

– дизайнери користувацьких додатків.

Основною зацікавленою стороною є власники користувацьких додатків. Вони прагнуть залучити та не втратити клієнтів за рахунок покриття як можна більшої частини ринку електронних девайсів. Також, вони прагнуть зменшити кількість виробничих витрат на створення та підтримку користувацьких додатків. Зробити розробку користувацьких додатків більш гнучкою та швидкою для можливості проведення АБ-тестувань, аналізу та швидкого прийняття рішень на користь бізнесу. Вони мають найбільший інтерес до вирішення поставленої проблеми.

Також, категорією зацікавлених сторін є проектні менеджери. Оскільки їм необхідно координувати та налагоджувати роботу платформо-орієнтованих команд.

Ще однією зацікавленою стороною є розробники мобільних додатків. Безпосередньо на їх роботу впливає наявність готового способу створення крос-платформних додатків.

Наступною категорією зацікавлених сторін є дизайнери користувацьких додатків. Їх інтерес полягає у можливості створювати однаковий дизайн, який не зорієнтований на особливості платформи, для однакової бізнес-логіки користувацьких додатків.

У табл. 7 зведено усі групи зацікавлених сторін, їх інтереси, та вплив (міра зацікавленості у вирішенні наявних проблем).

5.3. Комерційне рішення. Основні характеристики

Відповідно до вище зазначених проблем, можна описати кінцевий продукт, що має їх вирішувати. Даний програмний продукт буде реалізовано, як описано у попередніх розділах, у вигляді бібліотеки, консольної утиліти та шаблону додатка, що може бути встановлено з ресурсу JavaScript/NodeJS модулів NPM та скопійовано з публічного сервісу даних GitHub.

Таблиця 7

Зацікавлені сторони

Зацікавлена сторона	Інтерес зацікавленої сторони	Вплив зацікавленої сторони	Стратегії приваблення зацікавлених сторін
Власники користувацьких додатків	Лояльність клієнтів за рахунок підтримки додатків на як можна більшій кількості типів сучасних електронних девайсів	Високий	Виступи на конференціях та IT-форумах
Проектні менеджери	Зменшення кількості команд, що задіяні на одному проекті	Високий	Виступи на конференціях та IT-форумах Надання можливості користування технологією безкоштовно за визначеними умовами
Розробники користувацьких додатків	Готовий спосіб розроблення крос-платформних додатків	Середній	Виступи на конференціях та IT-форумах Надання можливості користування технологією безкоштовно за визначеними умовами
Дизайнери користувацьких додатків	Зняття платформних обмежень	Низький	Виступи на конференціях та IT-форумах

Завдяки їх взаємодії стало можливим, щоб веб-додаток працював із платформними API, які надають операційні системи смартфонів так, як з "рідними" мобільними додатками. І не втратити сумісність із веб-середовищем.

Зміни будуть помітними у більшій мірі для розробників та бізнесу. Тому що зменшиться час та витрати ресурсів на вирішення проблеми, визначеної у підрозділі 5.1.

Саме тому, клієнтом даного продукту є власники користувацьких додатків, а співпраця буде побудована на моделі співробітництва бізнесу для бізнесу, або як він ще називається B2B.

5.4. Конкурентні переваги рішення

Так як вже зазначалося вище, бізнеси зацікавлені у єдиній кодовій базі для різних платформ та у зменшенні виробничих витрат. Тому зацікавлені у використанні в розробці мобільних додатків таких технологій, як ReactNative, PhoneGap та Ionic.

Наразі існують методи та підходи використання WebView, що дозволяють створювати крос-платформні додатки. За рахунок цього можна пришвидшити розроблення та забезпечити гнучкість проекту. Але дані технології є виконуваними на обмеженій кількості платформ та потребують створення специфічних бібліотек для взаємодії із "рідними" сервісами операційних систем мобільних девайсів.

Отже, конкурентними перевагами програмного продукту, що пропонується, є:

– єдина кодова база;
– спрощена інтеграція із існуючими системами;
– легкість створення необхідних модулів для взаємодії із системними сервісами;
– можливість вже на даному етапі запустити систему на більше ніж 5-ти платформах: iOS, Android, веб, Ubuntu, tvOS, Windows, …

5.5. Клієнти. Сегменти ринку споживання

Мобільні додатки відкрили нові можливості для розробників і способи продажів для власників бізнесу. Вони приносять гроші там, де раніше потрібні були непідйомні інвестиції. Саме в цифрову економіку бюджети невеликих компаній і великих транснаціональних об'єднань конкурують на рівних.

Час – гроші, і в гик-економіці це головне правило. Користувачі мобільних додатків щодня витрачають на них по 3,5 години особистого часу. Тому інвестиції в рекламу додатків досягли 65% від всіх інтернет-комунікаційних технологій.

Якщо в минулому році торгівля через маркети додатків принесла $ 36,2 мільйона, то до 2020 ця цифра виросте майже в два рази – до $ 71,1 мільйона. А через два роки загальний ринок продажів через мобільні додатки взагалі складе $ 189 млрд.

Для досягнення максимальної ефективності діяльності команди проекту рекомендується провести сегментацію ринку клієнтів, тобто виділити більш-менш однорідні групи користувачів, зацікавлених в пропонованому продукті.

Таким чином, сегментацію ринку клієнтів для розроблюваної бібліотеки можна провести за двома категоріями:

– категорії мобільних додатків (рис. 19);

– можливість купити ліцензію на використання продукту;

– направленість компанії.

Згідно з цими двома критеріями, потенційних клієнтів можна розділити на 3 групи:

– Продуктові IT-компанії, продуктом яких є мобільний додаток в одній з категорій. Також вони готові заплатити гроші за ліцензію на використання продукту, тим самим зняти обмеження на доступний обсяг робіт і рівень їх вкладеності.

– Бізнеси, каналом доходу яких є мобільний додаток в одній з категорій. Також вони готові заплатити гроші за ліцензію на використання продукту, тим самим зняти обмеження на доступний обсяг робіт і рівень їх вкладеності.

– Розробники мобільних застосунків для власних, не комерційних, проектів. Вони не платитимуть за ліцензію.

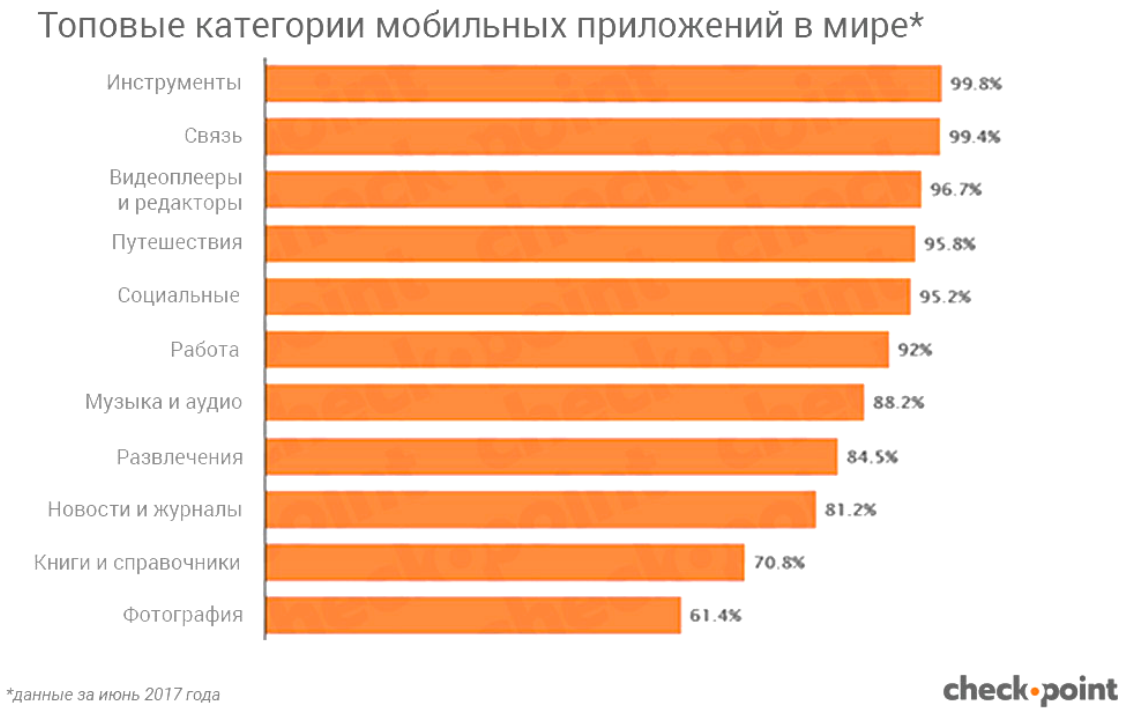


Рис. 19. Графік популярності мобільних додатків по категоріях

До першої категорії користувачів відносяться клієнти, каналами доходу яких є користувацькі застосунки – ця категорія включає в себе клієнтів, які є основним джерелом доходу для компанії.

До другої відносяться користувачі, які знайомляться з продуктом компанії та отримують його безкоштовно.

5.6. Унікальна ціннісна пропозиція

Ціннісна пропозиція – це пояснення того, як продукт вирішує проблему. Його можна скласти за формулою:

Ціннісна пропозиція = Проблема + Рішення / Продукт.

У дереві проблем було виділено низку проблем, а у зацікавлених сторонах – було визначено очікування відповідних сторін від продукту. Бізнеси бажають отримати швидкий користувацький додаток, що матиме гнучкість та швидкість внесення змін. Потребуватиме менше затрат ресурсів у вигляді часу та фінансів. Збільшить кількість підтримуваних сучасних платформ, що дозволить користувачам доступатися до даних з

улюблених зручних девайсів. Менеджери проектів – полегшення процесу розробки проекту за рахунок утримання лише однієї команди розробників. Розробники – швидке вирішення поставленої задачі.

Дійсно, запропоноване рішення, дозволяє частково задовольнити всі з наведених вище вимог зацікавлених сторін та вирішити наведені нижче проблеми.

Отже, основною унікальною ціннісною пропозицією є поєднання розроблених модулів у систему, що може бути використана для розроблення крос-платформних користувацьких додатків. А основними перевагами є виконання легкість інтеграції з існуючими платформами мобільних пристроїв.

5.7. Доходи та витрати

Дохід складається з продажу ліцензії на використання програмного забезпечення.

Продаж програмного забезпечення планується шляхом продажу ліцензії на програмне забезпечення. Слід зазначити, що ліцензію можна буде придбати за різними тарифними планами. Тарифи буде складено залежно від терміну використання та призначення цільового власницького програмного продукту: комерційний чи не комерційний.

Оцінюючи витрати на проект, варто виділити дві основні категорії: стартові і щомісячні витрати. Стартові витрати – це гроші, які вкладаються один раз на початку проекту, а щомісячні, відповідно, вкладаються в розвиток проекту щомісяця.

До стартових витрат відноситься:

– реклама (маркетинг);

– на юридичні послуги;

– на ріелтерські послуги, мінімальні меблі, канцелярію.

До щомісячних витрат відносяться:

– зарплати за посадами;

– оренда офісного приміщення;

– комунікації;

– податки;

– непередбачені витрати.

Детальніше ознайомитися з витратами на реалізацію проекту та прогнозованим прибутками можна з табл. 8.

Таблиця 8

Витрати на реалізацію проекту

Найменування витрат	1-й місяць, т. $	2-й місяць, т. $	3-й місяць, т. $	4-й місяць, т. $	5-й місяць, т. $	6-й місяць, т. $
Загальні витрати	7.5	1.5	1.5	1.3	1.7	1.2
Стартові витрати	7	1	1	0.5	0.5	
Реклама (маркетинг)	1	1	1	0.5	0.5	
Юридичні послуги	1					
Обладнання офісу	5					
Щомісячні витрати	0.5	0.5	0.5	0.8	1.2	1.2
ЗП				0.2	0.5	0.5
Оренда приміщення та комунікації	0.5	0.5	0.5	0.6	0.7	0.7
Заплановані прибутки			1	2	2	5
Результат (без оподаткування)	–7.5	–1.5	–0.5	0.7	0.3	3.8
Найменування витрат	7-й місяць, т. $	8-й місяць, т. $	9-й місяць, т. $	10-й місяць, т. $	11-й місяць, т. $	12-й місяць, т. $
Загальні витрати	1.2	1.2	1.2	1.2	1.2	1.2
Стартові витрати						
Реклама (маркетинг)						
Юридичні послуги						

Продовження табл. 8

Обладнання офісу						
Щомісячні витрати	1.2	1.2	1.2	1.2	1.2	1.2
ЗП	0.5	0.5	0.5	0.5	0.5	0.5
Оренда приміщення та комунікації	0.7	0.7	0.7	0.7	0.7	0.7
Заплановані прибутки	5	10	14	18	20	30
Результат (без оподаткування)	3.8	8.8	12.8	16.8	18.8	28.8

5.8. Бізнес-модель

Узагальнимо, все написане вище у лаконічну бізнес-модель у вигляді lean canvas.

Споживачі: бізнеси, що отримують дохід через користувацькі застосунки.

Проблема: висока вартість розробки системи з виконанням таких блоків: віддалений сервер, мобільний додаток, веб-додаток; утримання користувачів у мобільному додатку; лояльність клієнтів бізнесів.

Рішення: розроблення крос-платформних користувацьких додатків.

Унікальна ціннісна пропозиція: система, що дозволяє на основі однієї кодової бази створити платформо-орієнтовані додатки, що представлятимуть цільову бізнес-логіку.

Потоки доходів: доходи від продажу ліцензій; доходи від підтримки програмного забезпечення.

Структура витрат: утримання персоналу для надання технічної підтримки (виплати заробітних плат, соціальних виплат); утримання робочих місць для персоналу (оплата за оренду офісу та комунальні послуги); податкові витрати; оплата послуг юриста, бухгалтера, прибиральниці; витрати на маркетинг.

Також в канву бізнес-моделі включаються структурні блоки: прихована перевага (перевага, яку не можливо скопіювати або купити),

ключові метрики (основні показники, що вимірюються) та канали (шляхи до користувачів).

Канали: через платформи замовлення робіт з ПЗ; конференції; бізнес-тренінги; інші бізнеси.

Ключові метрики: кількість проданих ліцензій.

Прихована перевага: легка інтеграція з існуючими платформами; підтримка більше 5-ти платформ вже зараз.

Бізнес-модуль наведено у зведеному вигляді у табл. 9.

Таблиця 9

Канва бізнес-моделі

<table>
<tr><td>Проблема

висока вартість розробки системи

утримання користувачів у мобільному додатку

лояльність клієнтів</td><td>Рішення

створення крос-платформних додатків

використання бібліотеки, що пришвидшує розрахунки

Ключові метрики

кількість проданих ліцензій</td><td>Унікальна ціннісна пропозиція

система, що дозволяє на основі однієї кодової бази створити платформо-орієнтовані додатки, що представлятимуть цільову бізнес-логіку</td><td>Прихована перевага

легка інтеграція з існуючими платформами

підтримка більше 5-ти платформ вже зараз

Канали

через платформи замовлення робіт з ПЗ;
конференції;
бізнес-тренінги;
інші бізнеси</td><td>Споживачі

бізнеси, що отримують дохід через користувацькі додатки</td></tr>
<tr><td colspan="3">Структура витрат

утримання персоналу;
утримання робочих;
податкові витрати;
витрати на послуги юриста, бухгалтера
витрати на маркетинг</td><td colspan="2">Потоки доходів

доходи від продажу ліцензій;
доходи від підтримки програмного забезпечення</td></tr>
</table>

Отже, зважаючи на дані у табл. 9, можна зробити висновок, що запропонована бібліотека, що реалізує описаний у дисертацій метод пришвидшення розрахунків на стороні мобільних додатків та не блокування користувацького інтерфейсу, має перспективи у своїй подальшій реалізації. Звичайно, проведений аналіз не враховує всіх ризиків та факторів, таких як специфіка оподаткування у країні ведення бізнесу, проте навіть наявних досліджень достатньо, щоб прогнозувати комерційний успіх продукту та його окупаємось.

5.9. Висновки до розділу 5

У даному розділі було проведено аналіз поточної ситуації у сфері розроблення та підтримки користувацьких додатків, виявлено наявні проблеми та підсумовано їх у відповідному дереві проблем. Наряду з проблемами було виділено основні зацікавлені сторони у вирішенні існуючих недоліків, ступінь впливу даних сторін на вирішення проблем. Як наслідок було запропоновано комерційне рішення з конкурентними перевагами, що задовольняє інтереси зацікавлених осіб, та виділено унікальну ціннісну пропозицію запропонованого продукту. Було проведено аналіз майбутніх клієнтів, досліджено сегменти ринку споживання. Це дозволило спрогнозувати потенційні доходи та витрати на реалізацію продукту. У результаті була описана бізнес-модель, що обґрунтовує доцільність реалізації даного продукту та прогнозує його потенційну окупаємось та прибутковість в подальшому.

ВИСНОВКИ

Крос-платформний підхід до розробки користувацьких додатків є досить популярним на сьогоднішній день. Адже, дозволяє зменшити витрати ресурсів на розроблення та підтримку додатків. А надання користувачам можливості доступатися до своїх даних за допомогою улюблених електронних пристроїв підвищує лояльність клієнтів та дозволяє власникам користувацьких додатків утримувати велику кількість постійних клієнтів.

У даній магістерській дисертації було поставлено за мету дослідити існуючі підходи крос-платформної розробки додатків. А саме, способів взаємодії із системними сервісами операційних систем мобільних пристроїв та із компонентом операційних систем смартфонів WebView, що надає можливість відтворювати веб-вміст на цільових платформах. А результатом досліджень має бути розроблення способу використання WebView для створення крос-платформних користувацьких додатків.

У першому розділі було проаналізовано існуючі рішення для створення крос-платформних додатків. Виявлено їх переваги та недоліки. Досліджено методи досягнення крос-платформності. Для проведення аналізу було створено порівняльну таблицю. На основі досліджень та результатів аналізу було сформовано вимоги до розроблюваної системи.

Основними вимогами стали: забезпечення каналу комунікації між текстом програми користувацького додатка та системними сервісами платформ; стандартизація модулів взаємодії із системними API; використання React Native в якості базової технології.

У другому розділі даної магістерської дисертації було детально викладено рішення, що пропонується. Модифікація підходу полягає у тому, що у розроблюваній системі було використано допоміжний рівень у вигляді React Native. Також модифікацією можна вважати сприйняття веб-середовища, як ще однієї цільової платформи. Адже, взаємодія із платформо-орієнтованою частиною тексту програми відбувається у тому ж

процесі, а не через допоміжний канал комунікації. А така взаємодія має відмінності від стандартного підходу.

У третьому розділі було описано розроблені системні модулі. А саме, класи, що складають канал комунікації між сторонами веб та React Native; особливості використання платформного компонента відображення веб-вмісту сторінок – WebView; плагіни – архітектурні компоненти, що відповідають за стандартизацію API взаємодії тексту програми користувацького додатка із системними сервісами; класи архітектури для централізованого доступу до плагінів на обох сторонах системи: веб та React Native. Завдяки їх взаємодії стало можливим, щоб веб-додаток працював із платформними API, які надають операційні системи смартфонів, так, як з "рідними" мобільними додатками. Також не буде втрачено сумісність із веб-середовищем.

У четвертому розділі даної дисертації було описано методологію тестування для розроблюваної системи, проаналізовано результати роботи та визначено напрями для подальшого вдосконалення системи у майбутньому. За результати тестування можна заявити, що розроблена система не має помилок, які можуть призвести до виключних ситуацій при роботі користувача. Тому може вважатися готовою до залучення бета-тестувальників.

За результатами проведеної роботи в останньому розділі було сформовано та побудовано бізнес-модель кінцевого продукту, що описує ключові моменти в організації діяльності, пов’язаної з поширенням розробленої системи для створення крос-платформних користувацьких додатків.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Eisenman, B. Learning React Native [Text] / Bonnie Eisenman. — 2nd edition. — Sebastopol: O'Reilly Media, 2017. — 242 p.
2. Kirupa, Understanding WebViews [Електронний ресурс] / Kirupa. — Режим доступу: https://www.kirupa.com/apps/webview.htm. — Дата доступу: 3.12.2019.
3. WebKit Org., The WebKit Opensource Project [Електронний ресурс] / WebKit Org. — Режим доступу: https://webkit.org/project/. — Дата доступу: 3.12.2019.
4. Garbade, M. Native vs. cross-platform app development: pros and cons [Електронний ресурс] / Dr. Michael J. Garbade. — Режим доступу: https://codeburst.io/native-vs-cross-platform-app-development-pros-and-cons-49f397bb38ac. — Дата доступу: 3.11.2019.
5. LeRoux, B. PhoneGap Beliefs, Goals, and Philosophy [Електронний ресурс] / Brian LeRoux. — Режим доступу: https://phonegap.com/blog/2012/05/09/phonegap-beliefs-goals-and-philosophy/. — Дата доступу: 2.11.2019.
6. Trice, A. PhoneGap Explained Visually [Електронний ресурс] / Andrew Trice. — Режим доступу: https://phonegap.com/blog/2012/05/02/phonegap-explained-visually/. — Дата доступу: 2.11.2019.
7. Wargo, J. Apache Cordova 4 Programming (Mobile Programming) [Text] / John M. Wargo. — 1st edition. — Boston: Addison-Wesley Profeccional, 2015. — 560 p.
8. Griffith, C. Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova [Text] / Chris Griffith. — 1st edition. — Sebastopol: O'Reilly Media, 2017. — 292 p.

9. Ionic, Core Concepts [Електронний ресурс] / Ionic. — Режим доступу: https://ionicframework.com/docs/intro/concepts. — Дата доступу: 2.11.2019.
10. Flutter, Flutter for iOS developers [Електронний ресурс] / Flutter. — Режим доступу: https://flutter.dev/docs/get-started/flutter-for/ios-devs. — Дата доступу: 2.11.2019.
11. Napoli, M. Beginning Flutter: A Hands On Guide to App Development [Text] / Varco L. Napoli. — 1st edition. — Birmingham: Wrox, 2019. — 528 p.
12. Aggarwal, R. 10 top Programming Languages in 2019 for Businesses [Електронний ресурс] / Ruchika Singh Aggarwal. — Режим доступу: https://codeburst.io/10-top-programming-languages-in-2019-for-developers-a2921798d652. — Дата доступу: 3.11.2019.
13. Google Inc., Building web apps in WebView [Електронний ресурс] / Google Inc. — Режим доступу: https://developer.android.com/guide/webapps/webview. — Дата доступу: 3.11.2019.
14. Kumar, S. How React Native Works? [Електронний ресурс] / Saket Kumar. — Режим доступу: https://www.codementor.io/saketkumar95/how-react-native-works-mhjo4k6f3. — Дата доступу: 3.11.2019.
15. Buckler, C. JavaScript HTML5 Programming [Text] / Craig Buckler. — 2nd edition. — Sebastopol: O'Reilly Media, 2012. — 241 p.
16. Morley, M. JSON-RPC 2.0 Specification [Електронний ресурс] / Matt Morley. — Режим доступу: https://www.jsonrpc.org/specification. — Дата доступу: 17.11.2019.
17. Turskyi, V. MOLE-RPC [Електронний ресурс] / Viktor Turskyi. — Режим доступу: https://www.npmjs.com/package/mole-rpc. — Дата доступу: 20.11.2019.
18. Facebook, Communication between native and React Native [Електронний ресурс] / Facebook. — Режим доступу: https://faceboo

k.github.io/react-native/docs/communication-ios. — Дата доступу: 22.11.2019.

19. Vepsäläinen, J. SuviveJS – Webpack and React: From apprentice to master [Text] / Juho Vepsäläinen. — 2st edition. — Scotts Valley: CreateSpace Independent Publishing Platform, 2016. — 284 p.
20. Turskyi, V. Yet another JSON RPC Library? [Електронний ресурс] / Viktor Turskyi. — Режим доступу: https://docs.google.com/presentation/d/1NCmVJalBJp0Gliyc8KCdExHTl-xztDz3v50V4Y2k0bQ/edit#slide=id.g43ac3735fc_0_757. — Дата доступу: 22.11.2019.
21. Facebook, Out-of-Tree Platforms [Електронний ресурс] / Facebook. — Режим доступу: https://facebook.github.io/react-native/docs/out-of-tree-platforms. — Дата доступу: 30.11.2019.
22. Souders, S. Even Faster Web Sites: Performance Best Practices for Web Developers [Text] / Steve Souders. — 1st edition. — Sebastopol: O'Reilly Media, 2009. — 256 p.
23. Google Inc., About Android App Bundles [Електронний ресурс] / Google Inc. — Режим доступу: https://developer.android.com/guide/app-bundle. — Дата доступу: 30.11.2019.
24. ViroMedia, Overview [Електронний ресурс] / ViroMedia. — Режим доступу: https://docs.viromedia.com/docs/viro-platform-overview. — Дата доступу: 30.11.2019.
25. Gasston, P. The Modern Web: Multi-Device Web Development with HTML5, CSS3, and JavaScript [Text] / Peter Gasston. — 1nd edition. — San Francisco: No Starch Press, 2013. — 264 p.
26. Simpson, K. You Don't Know JS: Async & Performance [Text] / Kyle Simpson. — 1nd edition. — Sebastopol: O'Reilly Media, 2015. — 296 p.
27. Simpson, K. You Don't Know JS: Up & Going [Text] / Kyle Simpson. — 1nd edition. — Sebastopol: O'Reilly Media, 2015. — 88 p.

ДОДАТКИ

Додаток 1

Лістинг програми

Лістинг 1. styles.js

```
import { Platform, StyleSheet } from 'react-native';


const styles = StyleSheet.create({
  container: {
    flex: 1,
    ...Platform.select({
      ios: {
        backgroundColor: 'red'
      },
      android: {
        backgroundColor: 'green'
      },
      default: {
        // other platforms, web for example
        backgroundColor: 'blue'
      }
    })
  }
});
```

Лістинг 2. reducer.js

```
import produce from "immer"

const INITIAL_STATE = {}

const byId = (state = INITIAL_STATE, action) => {
    switch (action.type) {
        case RECEIVE_PRODUCTS:
            return {
                ...state,
                ...action.products.reduce((obj, product) => {
                    obj[product.id] = product
                    return obj
                }, {})
            }
        default:
            return state
    }
}

const bySecondId = produce((draft, action) => {
    switch (action.type) {
        case RECEIVE_PRODUCTS:
            action.products.forEach(product => {
                draft[product.id] = product
            });
            break;
    }
}, INITIAL_STATE)
```

Лістинг 3. DeviceInfo plugin (React Native)

```
import DeviceInfo from 'react-native-device-info';

const DEVICE_TYPE_MAPPING = {
    Handset : 'mobile',
    Tablet  : 'tablet',
    Tv      : 'tv',
    unknown : 'unknown'
};

class PluginDeviceInfo {

    async getUniqueID() {

        return DeviceInfo.getUniqueID();

    }

    async getAppName() {

        return DeviceInfo.getApplicationName();

    }

    async getBatteryLevel() {

        const level = await DeviceInfo.getBatteryLevel();

        return Number(level).toFixed(2);

    }


    async getBrand() {

        return DeviceInfo.getBrand();

    }


    async getBuildNumber() {

        return DeviceInfo.getBuildNumber();

    }


    async getCarrier() {

        return DeviceInfo.getCarrier();

    }
```

```
async getCountry() {
    return DeviceInfo.getDeviceCountry();
}

async getDeviceType() {
    const rnDeviceType = await DeviceInfo.getDeviceType();
    const deviceType = DEVICE_TYPE_MAPPING[rnDeviceType] || '';
    return deviceType;
}

 async getLocale() {
     return DeviceInfo.getDeviceLocale();
 }

 async getVersion() {
     return DeviceInfo.getReadableVersion();
 }
```

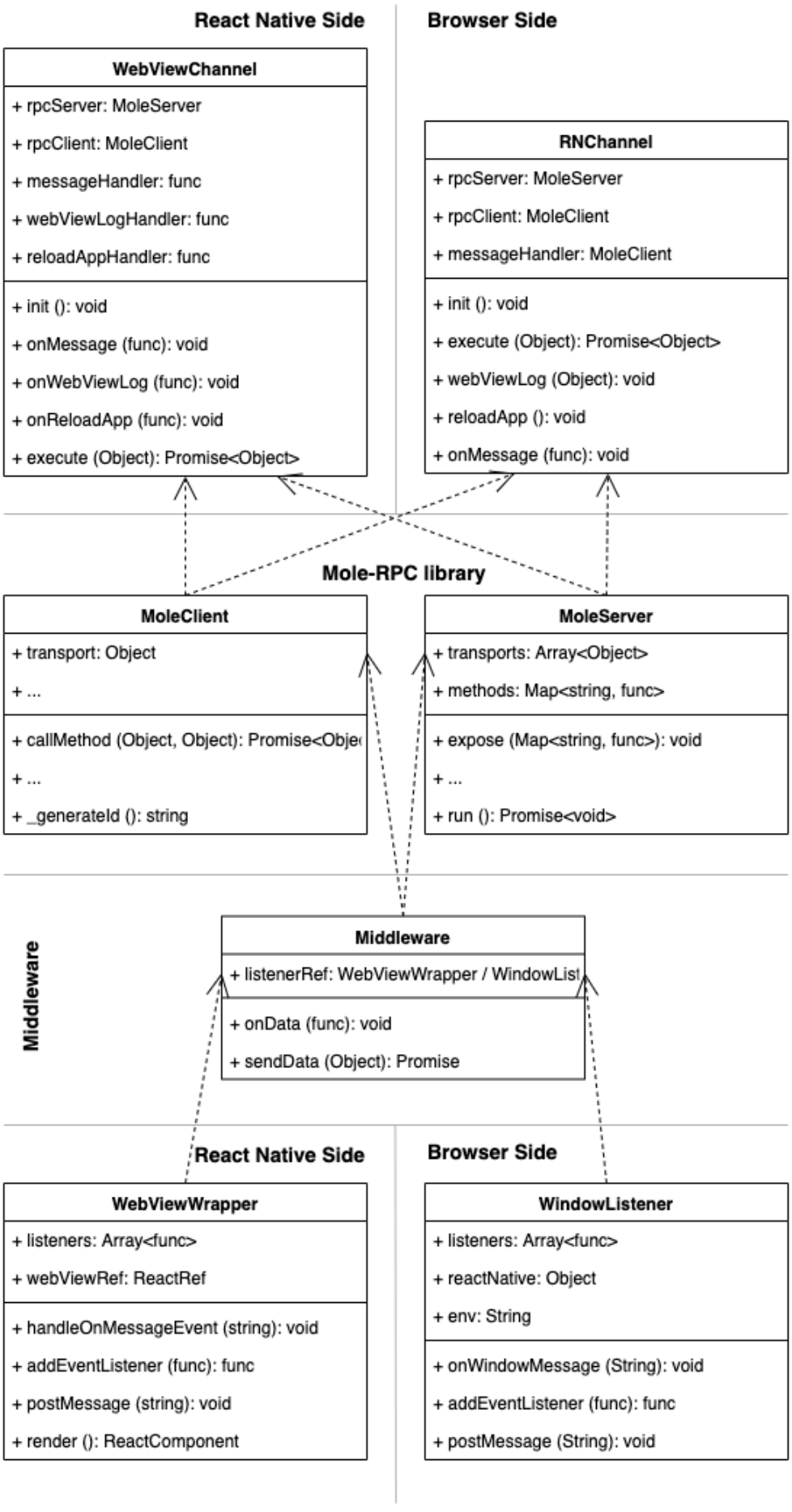
Лістинг 4. DeviceInfo plugin (веб)

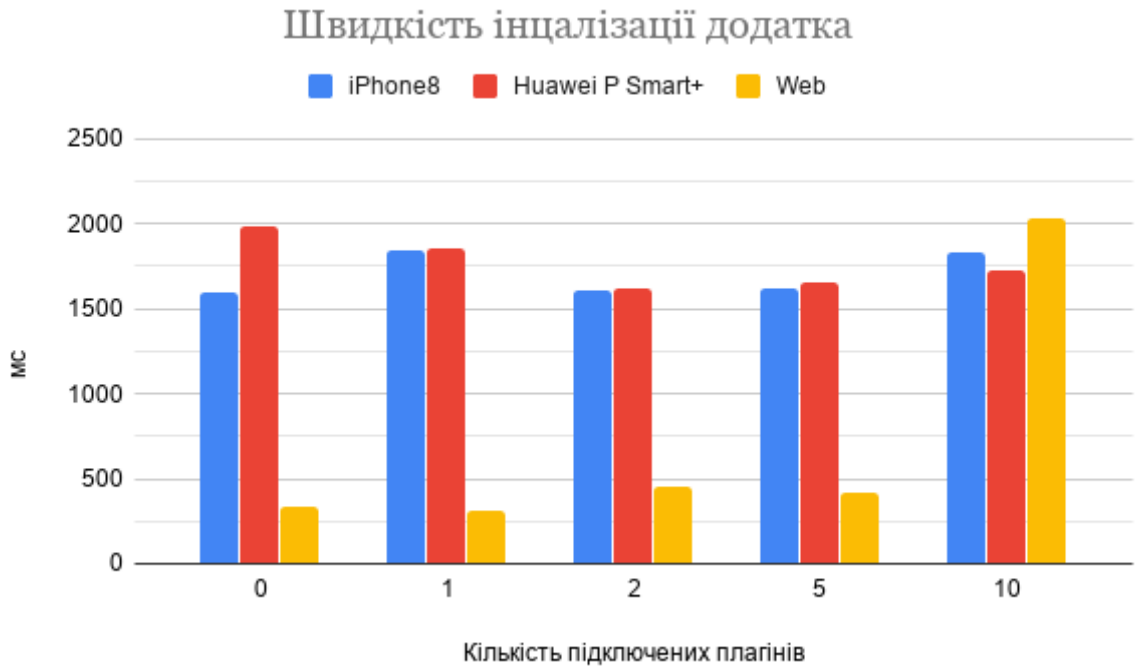
```
import UAParser from 'ua-parser-js';
import packageJson from 'package.json';

class DeviceInfo {
    constructor() {

        this._uaParser = new UAParser();

    }


    async getAppName() {

        return packageJson.name;

    }


    async getBuildNumber() {

        return 0;

    }


    async getDeviceType() {

        return 'browser';

    }


    async getBrowser() {

        return this._uaParser.getBrowser().name;

    }


    async getBrowserVersion() {

        return this._uaParser.getBrowser().version;

    }



    async getLocale() {

        return navigator.language;

    }
```

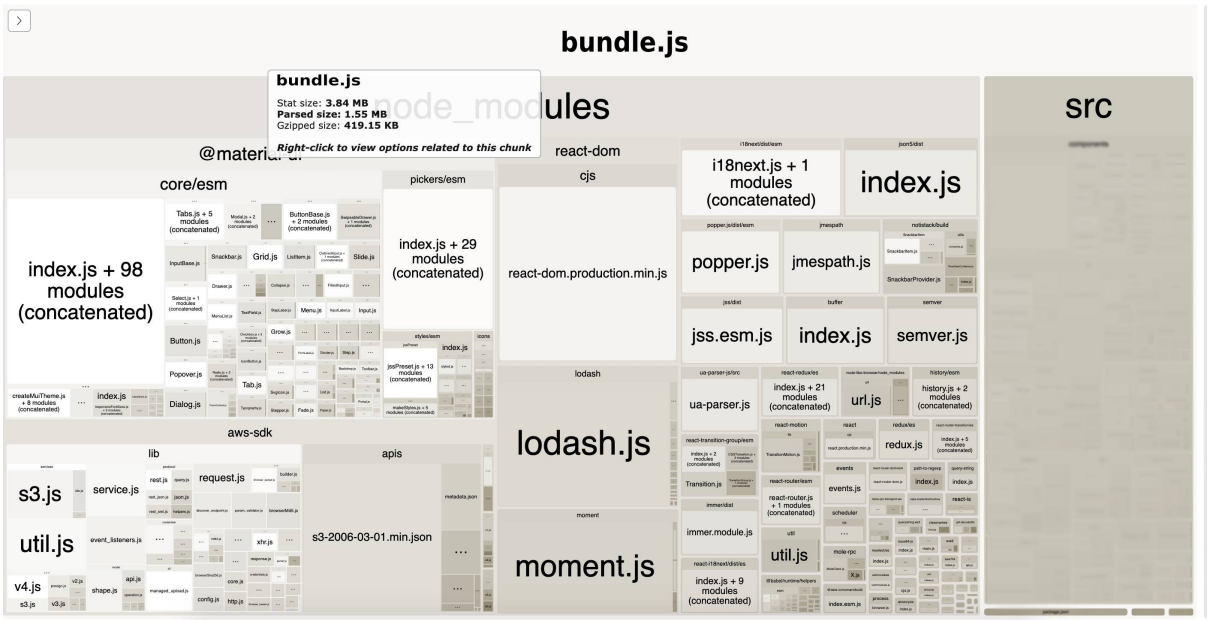
```
async getVersion() {
        return packageJson.version;
}

async getSystemName() {
        return this._uaParser.getOS().name;
}

async _getTimezone() {
        return '';
}
```

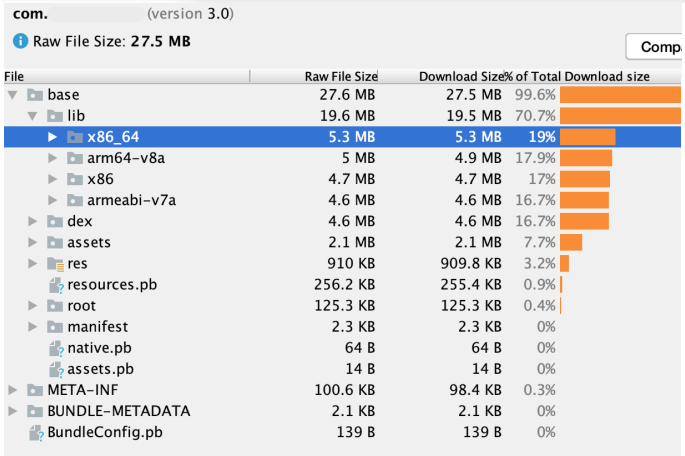
Додаток 2

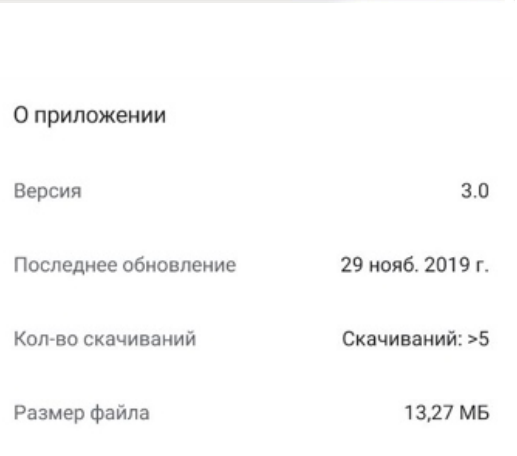
Копії графічних матеріалів

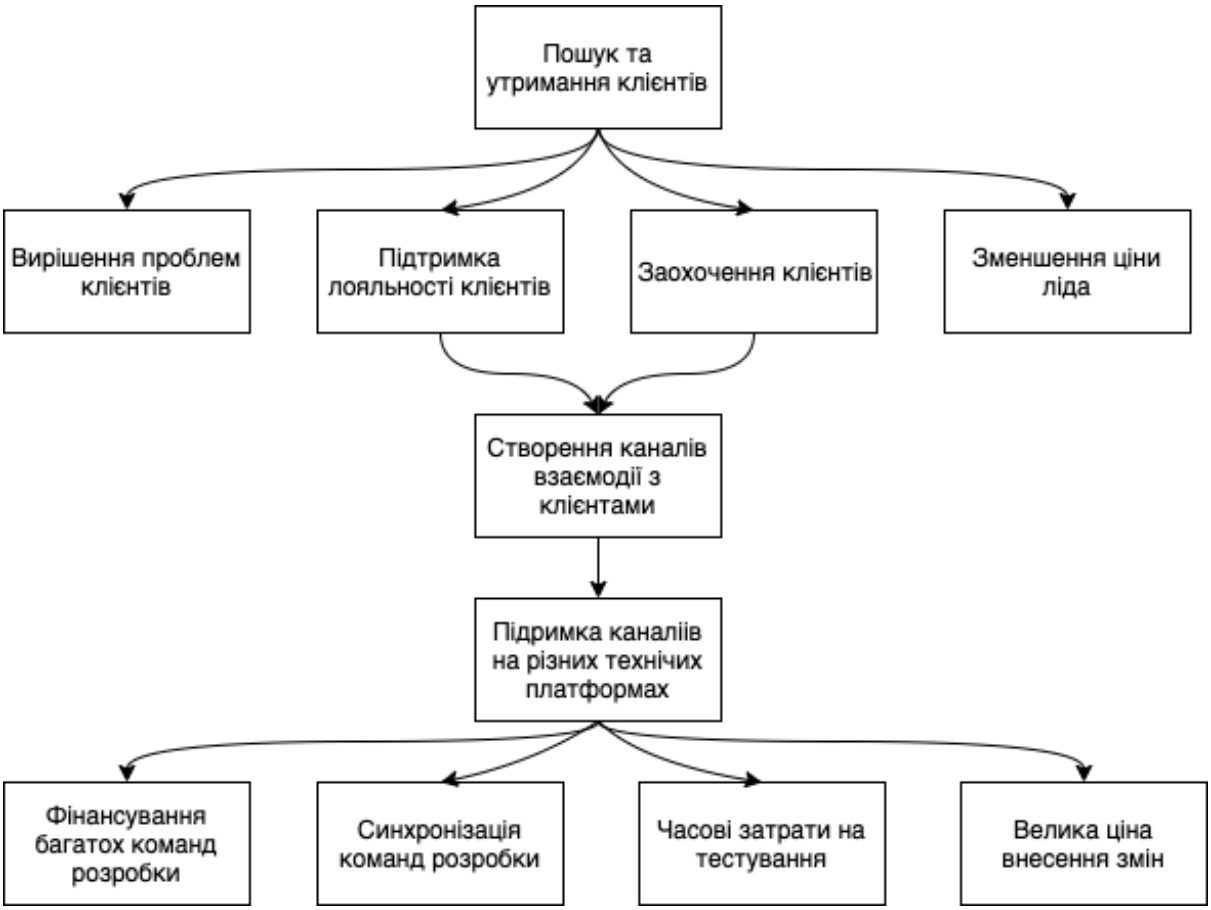

React Native Side
Browser Side
WebViewChannel
+ rpcServer: MoleServer
+ rpcClient: MoleClient
+ messageHandler: func
+ webViewLogHandler: func
+ reloadAppHandler: func
+ init (): void
+ onMessage (func): void
+ onWebViewLog (func): void
+ onReloadApp (func): void
+ execute (Object): Promise<Object>
RNChannel
+ rpcServer: MoleServer
+ rpcClient: MoleClient
+ messageHandler: MoleClient
+ init (): void
+ execute (Object): Promise<Object>
+ webViewLog (Object): void
+ reloadApp (): void
+ onMessage (func): void
Mole-RPC library
MoleClient
+ transport: Object
+ ...
+ callMethod (Object, Object): Promise<Obje
+ ...
+ _generateId (): string
MoleServer
+ transports: Array<Object>
+ methods: Map<string, func>
+ expose (Map<string, func>): void
+ ...
+ run (): Promise<void>
Middleware
Middleware
+ listenerRef: WebViewWrapper / WindowList
+ onData (func): void
+ sendData (Object): Promise
React Native Side
Browser Side
WebViewWrapper
+ listeners: Array<func>
+ webViewRef: ReactRef
+ handleOnMessageEvent (string): void
+ addEventListener (func): func
+ postMessage (string): void
+ render (): ReactComponent
WindowListener
+ listeners: Array<func>
+ reactNative: Object
+ env: String
+ onWindowMessage (String): void
+ addEventListener (func): func
+ postMessage (String): void

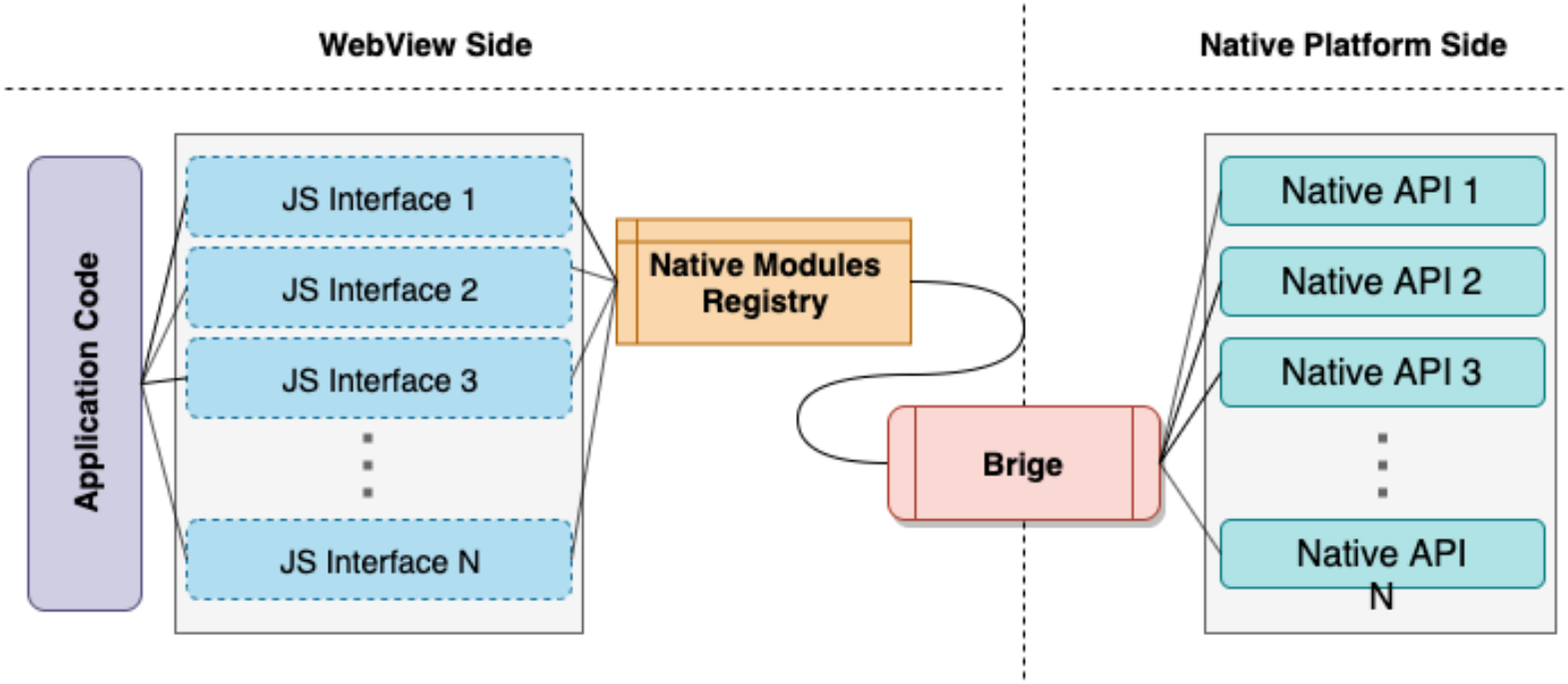

Швидкість інцалізації додатка
iPhone8
Huawei P Smart+
Web
2500
2000
1500
1000
500
0
мс
0
1
2
5
10
Кількість підключених плагінів

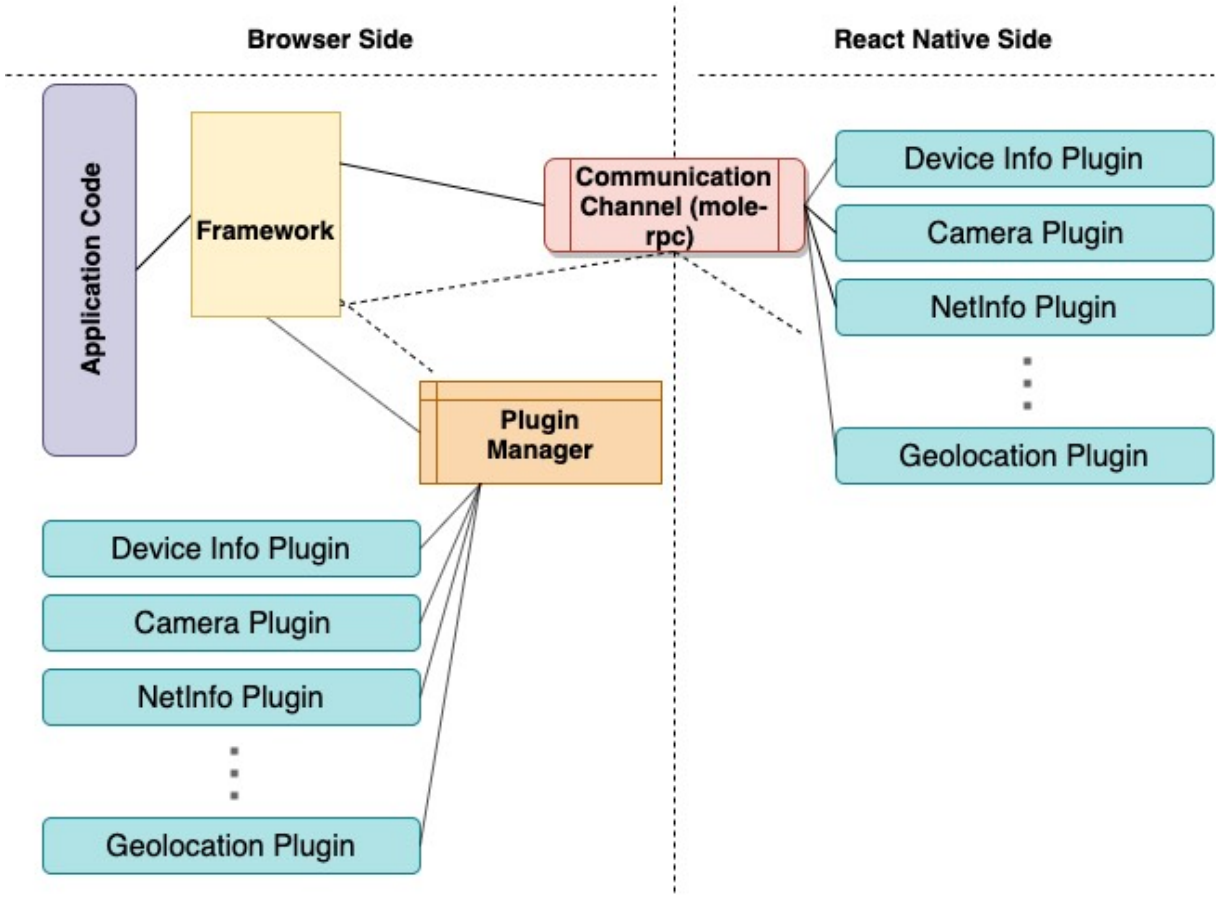

bundle.js
bundle.js
Stat size: 3.84 MB
Parsed size: 1.55 MB
Gzipped size: 419.15 KB
Right-click to view options related to this chunk
node_modules
src
@material-ui
core/esm
index.js + 98 modules (concatenated)
pickers/esm
index.js + 29 modules (concatenated)
react-dom
cjs
react-dom.production.min.js
i18next.js + 1 modules (concatenated)
index.js
popper.js
jmespath.js
jss.esm.js
index.js
semver.js
lodash
lodash.js
ua-parser.js
index.js + 21 modules (concatenated)
url.js
history.js + 2 modules (concatenated)
redux.js
events.js
util.js
immer.module.js
index.js + 9 modules (concatenated)
aws-sdk
lib
s3.js
service.js
request.js
util.js
v4.js
apis
s3-2006-03-01.min.json
moment
moment.js


com.
(version 3.0)
Raw File Size: 27.5 MB
Comp
File
Raw File Size
Download Size
% of Total Download size
base 27.6 MB 27.5 MB 99.6%
lib 19.6 MB 19.5 MB 70.7%
x86_64 5.3 MB 5.3 MB 19%
arm64-v8a 5 MB 4.9 MB 17.9%
x86 4.7 MB 4.7 MB 17%
armeabi-v7a 4.6 MB 4.6 MB 16.7%
dex 4.6 MB 4.6 MB 16.7%
assets 2.1 MB 2.1 MB 7.7%
res 910 KB 909.8 KB 3.2%
resources.pb 256.2 KB 255.4 KB 0.9%
root 125.3 KB 125.3 KB 0.4%
manifest 2.3 KB 2.3 KB 0%
native.pb 64 B 64 B 0%
assets.pb 14 B 14 B 0%
META-INF 100.6 KB 98.4 KB 0.3%
BUNDLE-METADATA 2.1 KB 2.1 KB 0%
BundleConfig.pb 139 B 139 B 0%


О приложении
Версия
3.0
Последнее обновление
29 нояб. 2019 г.
Кол-во скачиваний
Скачиваний: >5
Размер файла
13,27 МБ


Пошук та утримання клієнтів
Вирішення проблем клієнтів
Підтримка лояльності клієнтів
Заохочення клієнтів
Зменшення ціни ліда
Створення каналів взаємодії з клієнтами
Підримка каналіів на різних технічих платформах
Фінансування багатох команд розробки
Синхронізація команд розробки
Часові затрати на тестування
Велика ціна внесення змін


WebView Side
Native Platform Side
Application Code
JS Interface 1
JS Interface 2
JS Interface 3
JS Interface N
Native Modules Registry
Brige
Native API 1
Native API 2
Native API 3
Native API N


Browser Side
React Native Side
Application Code
Framework
Communication Channel (mole-rpc)
Plugin Manager
Device Info Plugin
Camera Plugin
NetInfo Plugin
Geolocation Plugin
Device Info Plugin
Camera Plugin
NetInfo Plugin
Geolocation Plugin

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

МОДИФІКОВАНИЙ ПІДХІД ВИКОРИСТАННЯ WEBVIEW ДЛЯ РОЗРОБЛЕННЯ МОБІЛЬНИХ ДОДАТКІВ

Виконала: Родіонова Віра Олексіївна

Науковий керівник: ст. викладач, к.нт.н., Люшенко Леся Анатоліївна

Київ - 2019

Проблема, актуальність, мета, завдання

Актуальність розроблення модифікованого підходу використання WebView

- Популярність мобільних додатків
- Різноманітність цільових платформ
- Популярність крос-платформної розробки
- Гнучкість процесу розроблення
  - Швидкість внесення змін
  - Кількість задіяних команд

«Якщо через декілька років вашого бізнесу не буде на екрані смартфону, то у вас не буде бізнесу» - Євген Черняк

Проблеми

- Різні цільові платформи (web, iOS, Android)
  - 3 кодові бази
  - 3 команди (із вузькою спеціалізацією)
    - Синхронізація
    - Час
    - Грошові ресурси

Проблеми крос-платформних технологій (продовження)

- Існуючі рішення
  - Обмеженість технологіями
  - Обмежений доступ до Native API (лише через сторонні бібліотеки)
  - Прозорість коду
- Наявність платформо-орієнтованого коду

Об’єкт та предмет дослідження, мета

Об’єкт дослідження: фреймворки для створення крос-платформних користувацьких додатків

Предмет дослідження: способи інтеграції з Native API мобільних платформ та архітектура фреймворків

Мета дослідження: модифікувати підхід використання WebView для розроблення крос-платформних користувацьких додатків

Завдання

Наукове завдання: дослідити способи інтеграції із Native API, створити канал комунікації між текстом програми користувацького додатку та Native API.

Окремі завдання

1. Проаналізувати існуючі фреймворки для створення крос-платформних користувацьких додатків
2. Модифікувати спосіб використання WebView для розроблення користувацьких додатків
3. Побудувати модель розроблюваного програмного забезпечення
4. Налаштувати взаємодію між WebView та React Native
5. Стандартизувати роботу із платформно-залежним кодом
6. Описати компоненти розроблюваної системи

Аналіз та порівняння існуючих рішень

PhoneGap / Cordova

APACHE CORDOVA™

2008 рік

3914 розробників

JavaScript

6 платформ

Середній рівень покриття Native API

Будь-які веб-технології

Ionic / Capacitor

2012 рік

341 розробник

JavaScript

4 платформи

Середній рівень покриття Native API

Angular, React, Vue

React Native

2015 рік

2038 розробників

JavaScript

2 платформи

Високий рівень покриття Native API

React

Існуючі рішення (порівняння)

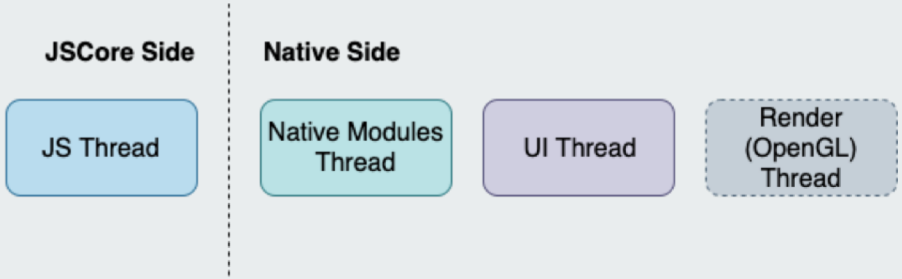
Критерій	Cordova	Ionic	React Native
Рік релізу / Стабільність	2008 / +	2012 / +	2015 / +
Спільнота	3914	341	2038
Мова JavaScript	+	+	+
Платформи	6	4	2
Легкість написання “плагінів”	-	- / +	+
Рівень покриття Native API	+ / -	+ / -	+
Модульність	+ / -	+ / -	+
Веб-технології	всі	Angular, React, Vue	React

Модифікований підхід до використання WebView для розроблення користувацьких додатків

Етап 1

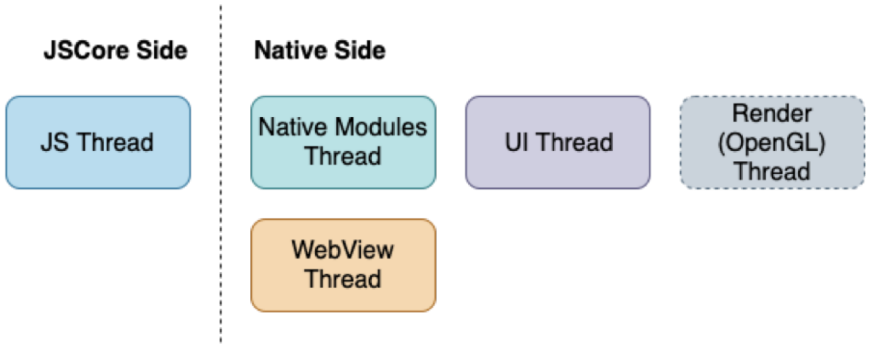
Проблема

Виконання тексту програми в окремому потоці



Рішення

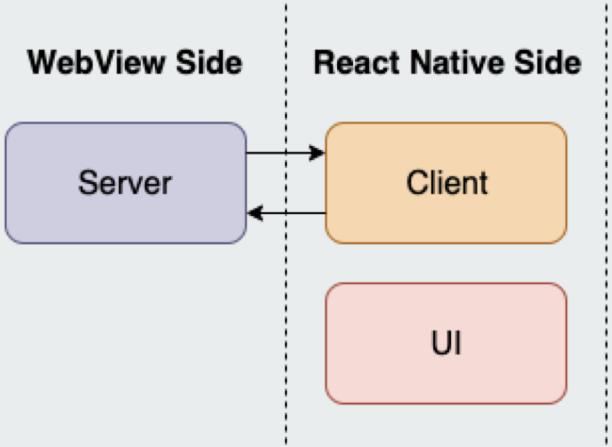
Використання WebView



Етап 2

Проблема

Комунікація між сторонами WebView та React Native



Рішення

Використання JSON-RPC 2.0

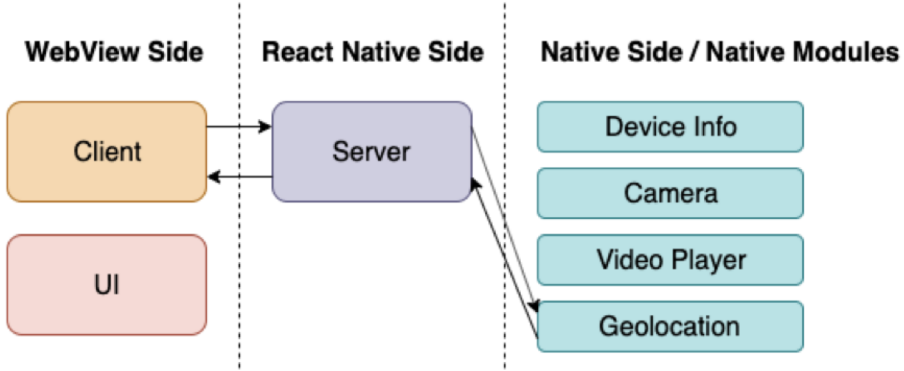
Етап 3

Проблема

Виконання тексту програми у трьох середовищах: **iOS**, **Android** та **web**. Доступ до Native API

Рішення

Відображення UI на стороні WebView



Етап 4

Проблема

WebView має обмеження щодо використання браузерного API

Рішення

Web як окрема платформа. Виділення абстракції для доступу до Native API - **плагінів**

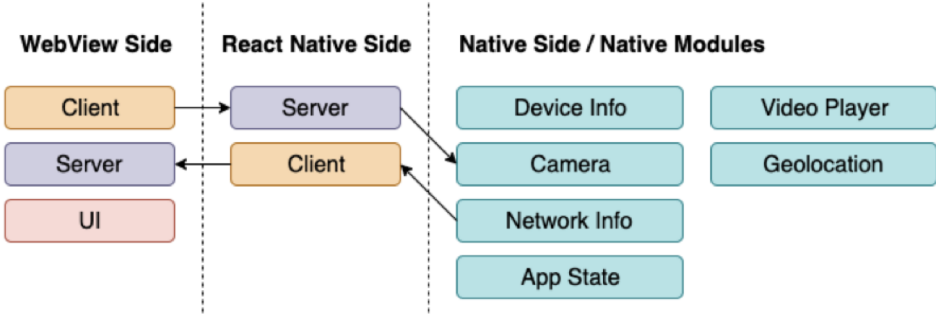
Етап 5

Проблема

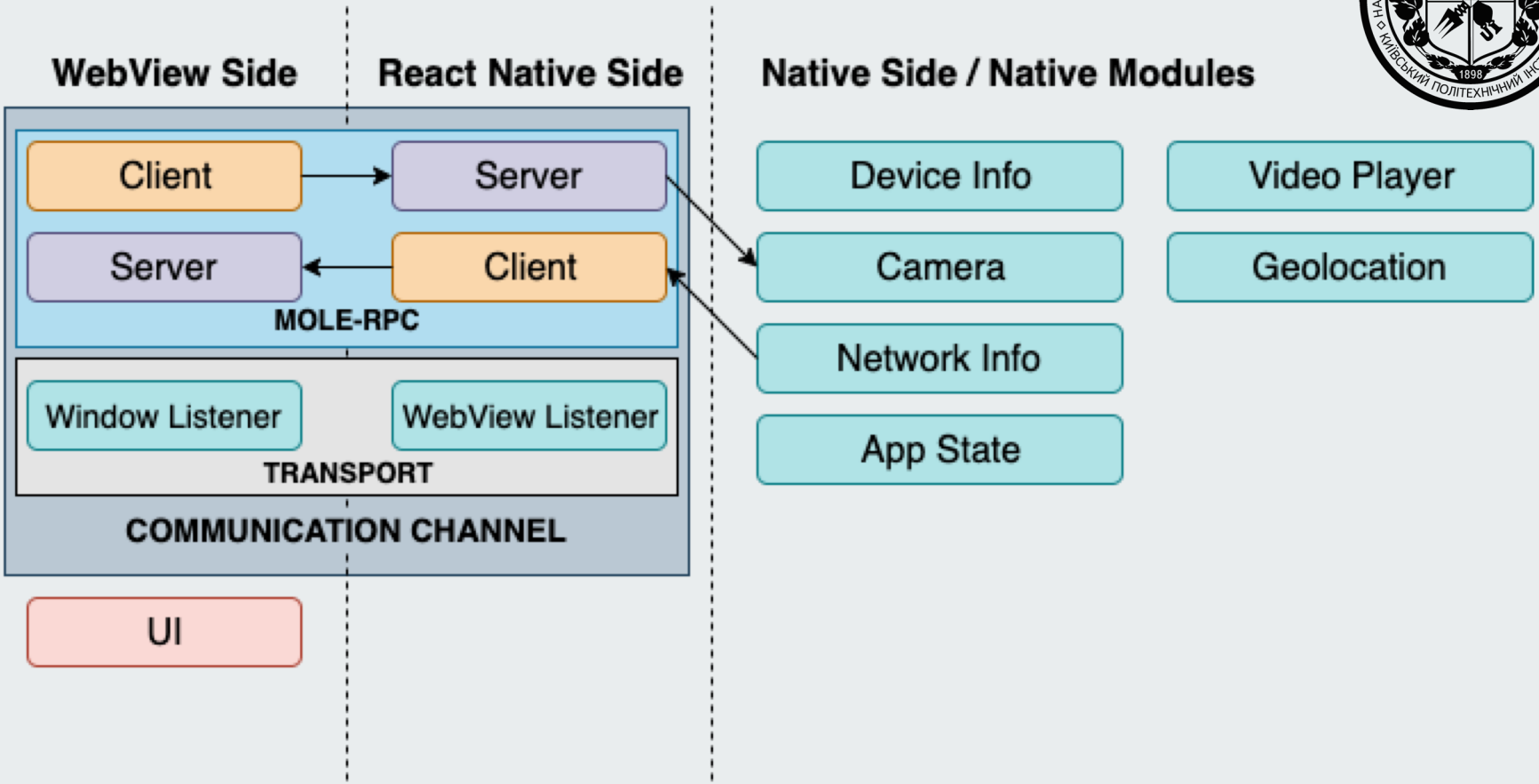
Реагування на стороні WebView на події від системних сервісів

Рішення

Двонаправлений канал комунікації. Інтеграція із mole-rpc (transport middleware)



Архітектура каналу комунікації додатка

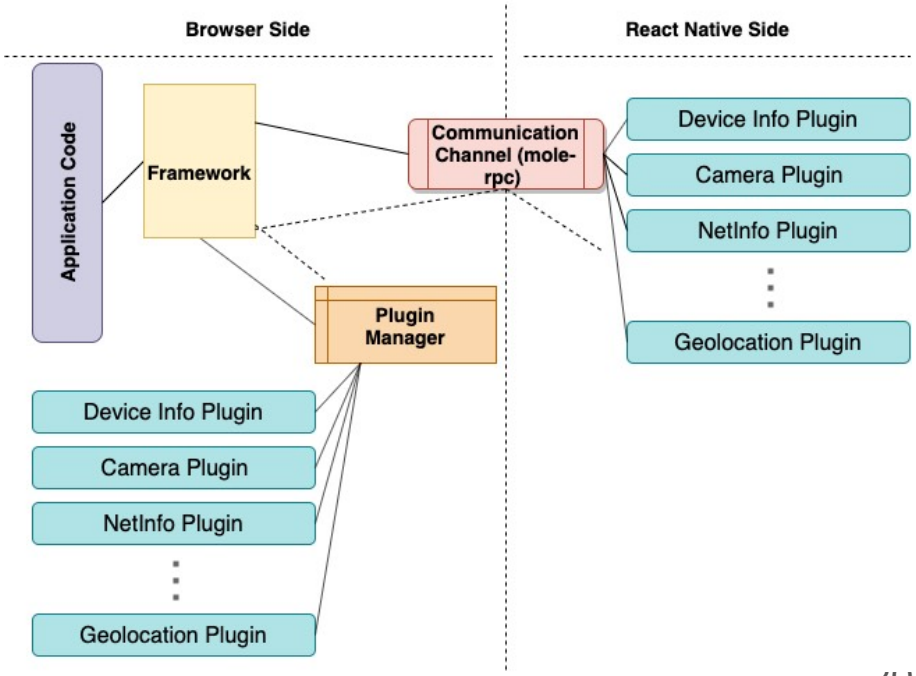


Етап 6

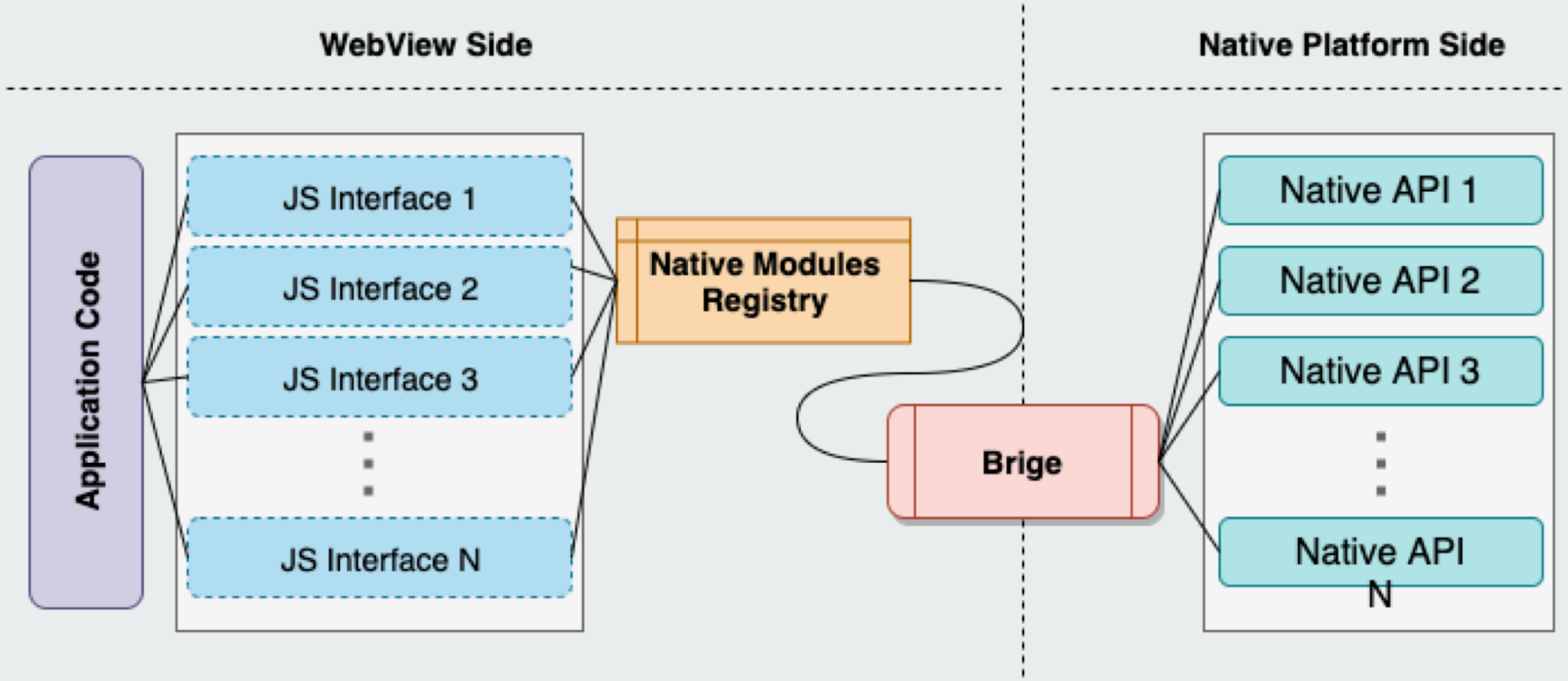
Проблема

Організація платформо-залежного тексту програми

Рішення



Загальна архітектура підходу



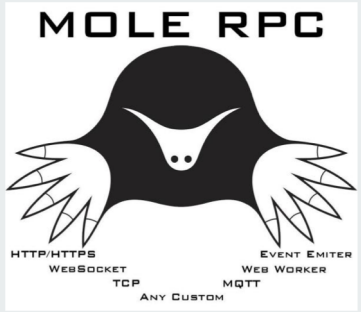
Приклади “плагінів”

	Утиліта	**Компонент**
Платформо-орієнтовані	LocalStorage DeviceInfo NetInfo	VideoPlayer DatetimePicker Camera
Платформо-незалежні	Progress Tracker Business logic plugins...	Message Bar Business logic plugins...

Аналіз розробленого програмного забезпечення

Використані технології

- React Native
- JSON RPC
- Mole RPC
- WebView



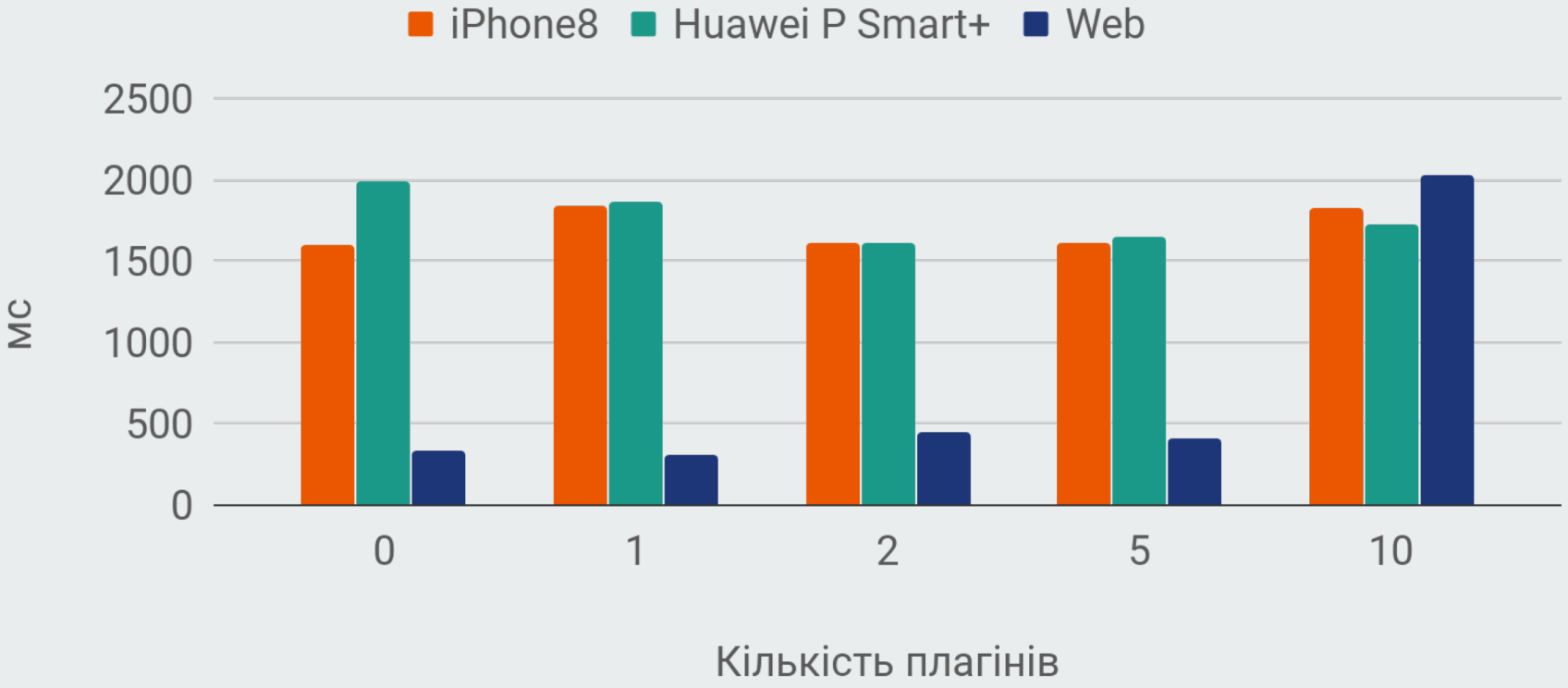
Тестування

3) Системне
Функціональне тестування методом «білого ящика»
Нефункціональне тестування: тестування на сумісність, стресове тестування

1) Модульне
Автоматизовані unit-тести для плагінів-утиліт кожної з платформ, mole-rpc middleware
Ручне тестування для плагінів-компонент

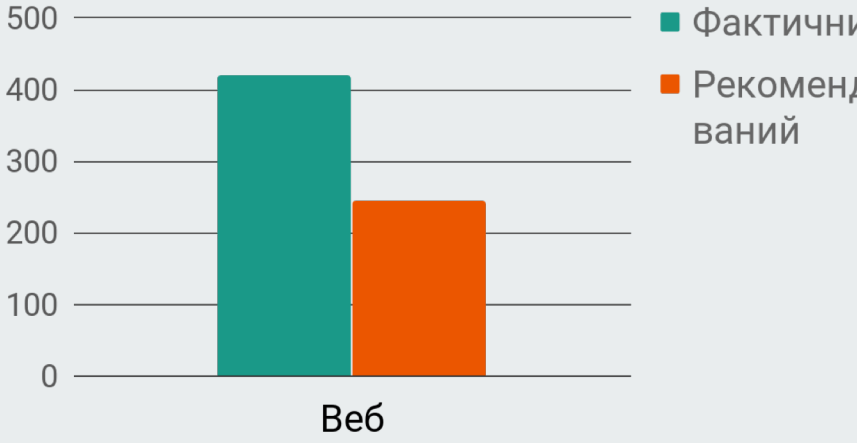
2) Інтеграційне
Взаємодія сторін React Native та веб
Підхід підходом «знизу вверх»

Швидкість ініціації додатка

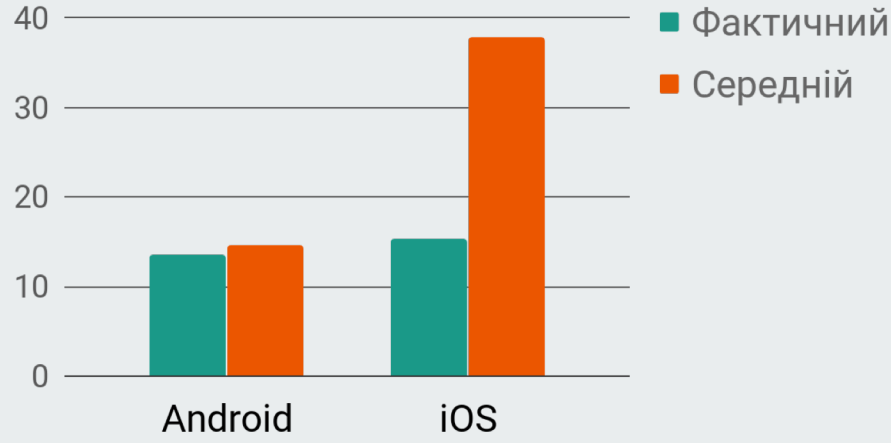


Розміри збірки

Web



Android / iOS



Плани на майбутнє / Вдосконалення

- Оптимізувати розміри збірок
- Опублікувати mole-rpc middleware у вигляді бібліотеки на NPM
- Опублікувати систему у вигляді шаблону на GitHub
- Адаптувати до інших платформ (Windows Phone, Ubuntu, macOS, tvOS)
- VR / AR модуль

Науково-інноваційна новизна

- Розроблено канал комунікації між React Native та WebView
- Стандартизовано взаємодію із системними сервісами та розроблено плагіни, що забезпечують доступ до платформо-залежних API
- Спільна кодова база для додатків на більш ніж 5 цільових платформ.


НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
1898

Висновки

Висновки

1. Проаналізовано фреймворки для створення крос-платформних користувацьких додатків

2. Описано загальний підхід взаємодії із Native API

3. Налаштовано взаємодію між WebView та React Native

4. Стандартизовано роботу із платформно-залежним кодом

5. Описано компоненти модифікованого підходу використання WebView створення крос-платформних користувацьких додатків

Висновки (продовження)

6. Розроблено програмну реалізацію модифікованого підходу використання WebView створення крос-платформних користувацьких додатків

7. Протестовано програмну реалізацію модифікованого підходу використання WebView створення крос-платформних користувацьких додатків

8. Проаналізовано ефективність програмної реалізації за різними критеріями

9. Сформовано напрями вдосконалення розробленого підходу використання WebView створення крос-платформних користувацьких додатків

Унікальність

UNICHECK

8.56% Matches

Highest match: **3.27%** with source **https://ela.kpi.ua/bitstream/**

5.86% Internet Matches	15
8.1% Library matches	119

0.27% Quotes

Quotes	1

No references found

0% Exclusions

No exclusions found

Replacement

No replaced characters found

Апробування отриманих результатів

XII наукова конференція магістрантів та аспірантів «Прикладна математика та комп’ютинг» (ПМК-2019).

ДЯКУЮ ЗА УВАГУ!